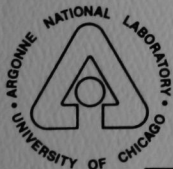


PROPERTY OF
ANL-W Technical Library

THE SYSTEMS ANALYSIS LANGUAGE TRANSLATOR (SALT): PROGRAMMER'S GUIDE

by

Howard K. Geyer and Gregory F. Berry



FOSSIL ENERGY PROGRAM

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

Operated by THE UNIVERSITY OF CHICAGO

for the U. S. DEPARTMENT OF ENERGY

under Contract W-31-109-Eng-38

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States government, and operated by The University of Chicago under the provisions of a contract with the Department of Energy.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

NTIS price codes
Printed copy: A08
Microfiche copy: A01

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue, Argonne, Illinois 60439

ANL/FE-85-4

THE SYSTEMS ANALYSIS LANGUAGE TRANSLATOR (SALT):
PROGRAMMER'S GUIDE

by

Howard K. Geyer* and Gregory F. Berry

Energy and Environmental Systems Division

March 1985

work sponsored by

U.S. DEPARTMENT OF ENERGY
Morgantown Energy Technology Center

*Engineering Division.

CONTENTS

ABSTRACT	1
1 INTRODUCTION	1
2 STRUCTURE OF THE SALT CODE	3
2.1 Salt Preprocessor	3
2.2 Driver Code	4
2.3 Model Structure	5
2.4 System Models	6
2.5 Properties Codes	7
3 ADDITION OF NEW MODELS AND FLOW TYPES	10
3.1 INTF File	10
3.2 Addition of New Models	12
3.3 Addition of New Flow Types	13
4 NUMERICAL PROCEDURES	15
4.1 Procedure for Using SOV	15
4.2 Procedure for Using SOLVG	16
4.3 Procedure for Using OPT	17
APPENDIX A: Code Listing	19
A.1 Combustor Model	21
A.2 Compressor Model	26
A.3 Deaerator Model	29
A.4 Gas-Diffuser Model	31
A.5 Fuel-Dryer Model	33
A.6 Feedwater-Heater Model	35
A.7 Flash-Tank Model	44
A.8 Gas-Turbine Model	46
A.9 Heater Model	50
A.10 Heat-Exchanger Model	52
A.11 Flow-Initiator Model	59
A.12 Fuel-Flow-Initiator Model	63
A.13 Molten-Carbonate Fuel-Cell Model	64
A.14 Liquid-Metal Diffuser Model	71
A.15 Magnetohydrodynamic-Generator Model	73
A.16 Liquid-Metal Magnetohydrodynamic-Generator Model	80
A.17 Liquid-Metal Nozzle Model	84
A.18 Liquid-Metal Pipe Model	86
A.19 Flow-Mixer Model	88
A.20 Gas-Nozzle Model	91
A.21 Phosphoric Acid Fuel-Cell Model	93
A.22 Pump Model	99
A.23 Pipe-Calculator Model	100
A.24 Steam-Condenser Model	101
A.25 Steam-Drum Model	103

CONTENTS (Cont'd)

A.26	Liquid-Gas Separator Model	105
A.27	Stack Model	110
A.28	Solid-Oxide Fuel-Cell Model	112
A.29	Flow-Splitter Model	117
A.30	Steam-Turbine Model	122
A.31	System Model	127
A.32	Two-Phase Diffuser Model	131
A.33	Two-Phase Mixer Model	135
A.34	Two-Phase Nozzle Model	137
A.35	General Properties Code	141

APPENDIX B: Job-Control Language for IBM System at ANL	145
--	-----

THE SYSTEMS ANALYSIS LANGUAGE TRANSLATOR (SALT): PROGRAMMER'S GUIDE

by

Howard K. Geyer and Gregory F. Berry

ABSTRACT

The Systems Analysis Language Translator (SALT), a systems-analysis and process-simulation computer code for steady-state and dynamic systems, can also be used for optimization and sensitivity studies. The SALT code uses sophisticated numerical techniques, including a hybrid steepest-descent/quasi-Newtonian multidimensional nonlinear equation solver, sequential quadratic programming methods as optimizers, and multistep integration methods for both stiff and nonstiff systems or equations. Based on a preprocessor concept, the code uses precompiled component models, several flow types, and numerous thermodynamic and transport property routines. The SALT code has been used to study open-cycle and liquid-metal magneto-hydrodynamic systems, fuel cells, ocean thermal energy conversion, municipal-solid-waste processing, fusion, breeder reactors, and geothermal and solar-energy systems. This programmer's guide briefly describes the code, defines parameters used with the component models, and presents detailed examples of declaration structures to be used with these models.

1 INTRODUCTION

The objective of the Systems Analysis Language Translator (SALT) programmer's guide is to describe in detail how the SALT code works, what is being calculated, and what a programmer needs to consider to add or modify component models, property routines, or solution procedures. Using this guide, a programmer should be able to add to or modify the computer coding to customize the routines for particular applications. The SALT code can be used to solve almost any systems-analysis problem -- possible applications include pure economics, electrical circuits, transportation, or even weapon deployment. (Some of these applications have actually been implemented, but they are not discussed here because they are not supported by the U.S. Department of Energy.) It is through an appreciation of this flexibility that the user can assess the value of SALT for a given application.

The SALT code is based on a preprocessor concept, wherein a "new" system driver can be written for each application. The generation and compilation of this system driver requires extremely small amounts of computer time, so it is not usually worthwhile to save the system driver for further applications on the same system. This efficiency in creating a system driver may be attributed to the fact that most of the

algorithms used for calculations are contained in a load module. Thus, all of the models, property routines, and solution procedures have been precompiled. Duplication of models (e.g., use of several mixers) presents no difficulty, because each model uses exactly the same algorithms and logic. The different data requirements for each model are distinguished by means of unique labels. This approach is consistent with the overall concept of creating an efficient and compact systems code.

Chapter 2 of this report describes the structural requirements for modifying or adding models or property routines to SALT. Understanding the structure is essential if the new models or property routines developed are to be consistent with existing models and property routines. This chapter also indicates what the developers consider to be mandatory requirements. The outputs of the model or property routines must be single-valued, returning unique and repeatable values for identical input flow-stream values and identical parameter values. Consistent property values should be contained in each output flow stream to ensure that this flow stream is consistent when it is put into a subsequent model. The account in Chapter 2 contains all the necessary structural information required to add to or modify any part of the models.

Chapter 3 provides the detailed information that must be known in order to add new models and new flow types. Probably most modifications to SALT will involve the addition of new models; therefore, a separate section is included to describe this procedure. Because of budget limitations, a section on adding new property routines and solution procedures has not been included. (Interested users may contact the authors if these modifications are required.) Work is in progress to develop additional models and new property routines (especially designed for the solution of chemistry problems).

Chapter 4 briefly describes the calling sequences to the solution procedures SOV, SOLVG, and OPT, furnished with SALT. The theoretical basis for these procedures is not part of the SALT documentation and is beyond the scope of either the user's guide* or the programmer's guide. This omission is consistent with our policy of not presenting the theoretical basis for the algorithms used in the models and property routines.

The integrated solution procedure also is not presented. This procedure is used predominantly with the dynamic models, which are not covered in detail in this documentation (again due to budget limitations). The integrated solution procedure also is used with the distributive-parameter models. However, these models are few; most of the models are lumped-parameter models.

Appendix A contains actual program listings (with comments) of some of the more frequently used models at Argonne National Laboratory (ANL). Appendix B gives the job-control-language procedure used. The property routines and solution procedures, less frequently used models, and dynamic models are not listed in this report. These routines are not thoroughly supported with comments, so their inclusion in the document was deemed inappropriate. These omitted listings are on the computer tape, which can be obtained directly from the National Energy Software Center, located at ANL.

*Geyer, H.K., and G.F. Berry, *The Systems Analysis Language Translator (SALT): User's Guide*, Argonne National Laboratory Report ANL/FE-85-03 (March 1985).

2 STRUCTURE OF THE SALT CODE

2.1 SALT PREPROCESSOR

Basically, SALT makes use of a preprocessor technique; the code reads the STRUCT file, consisting of the system problem under consideration, and then generates a driver code that will solve that problem. The driver code, written in PL/I, is then compiled and run in order to perform the system analysis.

In generating the driver code, the SALT code parses the input STRUCT file to determine which models are being called and what iterative tasks have been defined. In order to establish an interface with the models, the SALT code also reads the interface file (INTF) to obtain all the variables that pertain to the models being used. The STRUCT file and INTF file are the only two input files that the SALT preprocessor requires. The output driver code is generated on the file SYSDRV. Any messages, together with a reflection of the input STRUCT file, are printed out on the SYSPRINT file.

The details of parsing by the SALT code are not explained here; however, the requirements that the SALT code places on certain variable names are considered, because such variables may be generated by the model developer. The SALT code is a very general code; it handles arbitrary data structures representing the variables of the flows and models, arbitrary model calls representing the processing of such data structures, and the logic and calls to certain mathematical procedures. Very few specific requirements are imposed on these data structures and model calls by the SALT code. However, because SALT must work with these data structures and model calls, it must be able to manipulate their names.

The flows and model data are represented by PL/I aggregate variables; these variables may be of arbitrary structure, with one exception: the first second-level variable must be the variable "NAME," declared with the attributes "CHAR(16)." The existence of such an element of the data structure is an assumption built into the SALT code, which will use that particular variable to store the name of the flow or model as it is used within the STRUCT file. These names should not be inadvertently redefined within the model, because output from the system run could be adversely affected.

The SALT code itself places no further restrictions on the structuring of the flows and model data structures. (The naming of some of the models' substructures is restricted in the manner described below.) Almost any type and amount of data can be associated with the flows and models used by SALT.

The model calls representing the processing of the flows and model data have three requirements imposed by SALT:

1. Each model must have associated with it a data structure that is always the first argument to the model call (technically, a pointer to the data structure is passed).

2. The name of the model call must be the same as the name of the data structure, followed by one or more characters representing the type of entry to the model. As an example, for the heat-exchanger model (HX) there exists a PL/I data structure denoted HX (i.e., the one-level name is HX) and a procedure with multiple entries HXC, HXH, and HXOUT, each of which has as its first argument a pointer to the HX data structure. SALT actually takes the name of this pointer to be the model name as used within the STRUCT file. Thus, if the HX model had been called HX_1, then HX_1 would be declared as a pointer to the HX data structure containing the data for the HX_1 heat exchanger. Technically, since the name of a PL/I external procedure must have seven characters or fewer, this imposes a length restriction on the model-type name. However, this restriction is imposed by the PL/I compiler, not by SALT.
3. All other arguments to any model entry represent pointers to the flows required by that model. As in the case of the model pointers, SALT takes the names of these pointers to be the names of the flows used within the STRUCT file.

2.2 DRIVER CODE

The driver code is the output, on the file SYSDRV, from the SALT code preprocessor. This code is all that is constructed by SALT, and its structure is relatively simple. The SALT code does not construct any component models or properties procedures, so this driver code usually is not very large. (The largest part is usually a reflection of part of the INTF file.)

The driver code can be divided into five sections. The first section consists of those parts of the INTF file that represent models and flows pertaining to the system under consideration. For each model or flow type used in the system, the appropriate model or flow interface is copied from the INTF file to the driver coding. Statements that allocate storage and pointers to this storage for each model and flow data structure used by the system follow.

The second section of the driver code pertains to the construction of the linked lists used by the system models. This construction process consists of a series of calls to an internal procedure to allocate a new link for each model substructure used by the system models. More information on this list structure will be given below.

The third section of the driver code is basically the initialization of model variables or other variables from the DATA input statement. This is the last statement in the SALT input, but it comes before most of the other executable statements in the driver code.

The fourth section represents the model calls and iterative loops defined by the SALT code input on the file STRUCT. For each model specified, a call is made to the

appropriate model with the appropriate flow arguments. For iterative loops (i.e., SYSBEG/SYSEND loops), the driver code is expanded into a PL/I DO loop. Before the DO statement, all flows used in the loop that were also used before this loop are saved. The saved flow variables are used to reinitialize the flows after the DO statement. Depending on the type of task being performed within the loop -- VARY/CONS, SWEEP, MINI, etc. -- certain variables needed by the equation solver or optimizer may be declared and initialized before the DO statement.

At the end of the iterative loop, again depending on the task, the user-imposed constraints and objective function will be evaluated and a call made to the equation solver or optimizer.

The fifth section of the driver code consists of some PRINT statements to produce a summary of the iterations completed by the iterative tasks.

2.3 MODEL STRUCTURE

The models that are called by SALT can be composed of any type of PL/I coding, representing practically any level of modeling sophistication. The few requirements on the models needed to establish an interface with SALT have already been indicated. (The arguments to any model entry must be pointers to the model data structure [PL/I aggregate variable] and then to any flow data structures. Also, the model entries must be denoted by a common model-type name, followed by additional characters representing the particular model entry.)

The model developer dictates the order of the flow arguments, which should be such as to facilitate their use in the SALT input. Pass-through flows should precede input flows, which in turn should precede output flows.

Being written in PL/I, the models can make full use of any of the PL/I programming constructs -- IF - THEN - ELSE statements, DO WHILE statements, array assignments, etc. To some extent, the use of PL/I eliminates the need for a model simulation language. The models may also call FORTRAN subroutines to perform the modeling calculations.

In terms of the actual modeling, no restrictions are made. However, it is important to make sure that the model is a true function of only the input flows and model parameters; there should be no hidden variables that may take different values each time the model is called. Thus, if a model is called twice with exactly the same inputs, it should return exactly the same outputs. This requirement is imposed so that the SALT code can employ mathematical procedures that may be used to calculate gradients of model outputs using the method of finite differences.

Most models have certain common features, although these are not strictly necessary for their use within the SALT code. First, most of the models save the flows of a model within the model's data structure. These flows can then be used within constraints or objective functions. Also, the system models can gain access to these flows and print them out along with the other flows. Models that put in, lose, consume,

or produce power (other than through energy transfers between flows) have a POWER substructure within the model's data structure to save these power changes. These POWER substructures are referenced by the system model to calculate the net power produced, as well as the system efficiency. Most models have a PARM substructure containing the parameters pertaining to a particular model. Finally, most models have an output entry, which is used to print out the values saved within the PARM substructure.

Appendix A presents the actual coding representing the individual models (with comments included). Most of this coding is easily understandable. However, because many of the models' calculations depend on property calculations, Sec. 2.5 discusses the calling sequences to the properties codes.

2.4 SYSTEM MODELS

Although system models are in all respects similar to the other component models, the former may also make use of a linked list of variables representing information contained in other models. Thus, system models may be used to sum up component powers or print out all of the system flow variables. This linkage mechanism is discussed here to help modelers develop new system models as the need arises.

The linked list used to locate the various model substructures is developed by SALT using information in the INTF file. This file (discussed in Chapter 3) has several types of statements, one of which -- denoted as type 0 -- informs the SALT code that a linked list is required. The type-0 statement has the name of a model substructure and the name of a pointer to the head of the linked list of that substructure type.

To develop an actual linkage structure, one generates a sequence of PL/I structure variables of the following form:

```
1 LINK BASED,
  2 MP POINTER,
  2 LP POINTER,
  2 NP POINTER;
```

where MP points to the beginning of the model structure containing the particular substructure for which the linked list is being developed, LP points to the particular substructure, and NP points to the next LINK structure. The first LINK structure is pointed to by the pointer named in the type-0 statement in the INTF file. The last LINK structure has NP set to the NULL pointer.

As different substructure types are specified in the type-0 interface statements, different linked lists are generated, starting at their corresponding head pointers. Thus, the LINK.LP pointer will point to different structure types. The LINK.MP pointer is included so that the name of the model that the corresponding LINK.LP is accessing can also be obtained.

Sometimes it is necessary to have linked lists pointing to structures that have no common name. For example, the saved flow structures are saved within the models as FLC1, FLC2, FLH, etc. These names were chosen to facilitate the system model. If one specifies FL*, the SALT code will generate a linked list consisting of all the substructure names starting with FL and followed by the name of the model entry label and any additional characters. The use of the added entry label permits SALT to generate the linked list in the order that the model entries are used. Otherwise, the same LINK structure is used as before. Several LINKs may be allocated for each model, depending on the number of substructures within the model that must be linked.

2.5 PROPERTIES CODES

The thermodynamic properties codes constitute one of the most important parts of the system code. Many of the models actually reduce to simply calling the properties codes one or more times with different inputs. Thus, it is important to understand what these codes are doing in order to understand the models. At present, numerous properties codes are in existence, but most of these are accessed through a common calling program (GP).

The GP code actually has three entries -- GPIN, GP, and GPSAT. The GPIN entry is usually called within the SALT input (and thus requires a model label, such as GP_1:IN) and is used to initialize the gas properties code. The GP entry, which is usually called by the models, performs the property calculations. Also called by the models, GPSAT performs calculations of the saturation properties.

The calling sequence to GP consists of three arguments -- LABEL, declared as a CHAR(16) variable; FLOW, declared the same as the GAS structure; and SW, declared as a FIXED BIN(15) variable. The LABEL argument, which contains the name of the model calling the properties code, is used to obtain printouts during debugging runs. The FLOW argument represents the particular flow for which the properties are being calculated. The SW argument is a switch that takes the values 1, 2, or 3, telling GP which variables within FLOW are to be treated as inputs. When SW is equal to 1, the variables FLOW.PROP.TEMP (representing the flow's temperature) and FLOW.PROP.PRES (representing the pressure) are inputs. When SW is equal to 2, FLOW.PROP.ENTH (representing the flow's specific enthalpy) and FLOW.PROP.PRES are the inputs. When SW is 3, FLOW.PROP.ENTP (representing the flow's entropy) and FLOW.PROP.PRES are the inputs. The other thermodynamic variables of the flow are calculated as functions of these inputs. Thus, calling GP with SW equal to 2 will obtain FLOW.TEMP, FLOW.RHO, FLOW.ENTP, and FLOW.QUAL as outputs.

The calling sequence to GPSAT consists of LABEL and FLOW (as in GP) and three other FLOAT(16) variables. The first of these variables is the critical pressure for the flow, and the other two are the saturation liquid and vapor enthalpy values as functions of the flow's pressure.

The actual fluid code used when GP or GPSAT is called is determined by the value of the flow's ID parameter. At present, ID can be either H2O, GAS, LIQ, JAN, or THR. When ID is H2O, the water/steam properties code is called. When ID is GAS, a

simple 23-species chemical equilibrium code is called. When ID is either LIQ, JAN, or THR, the specific fluid name must follow these ID designators. These fluids must be among those in the files LIQDATA, JANDATA, or THRDATA, for the corresponding ID type.

The gas properties code can handle a variety of different gases. At present, information about the gas type is carried along with the flow variables in the substructures ATOM and COMP. The substructure COMP contains the molar fractions of the 23 species, while ATOM is an array representing the kilogram-atoms per kilogram of flow of the eight elements that make up these 23 species. The gas properties procedures also include GASBW and GASNM, which are sometimes called directly by the models (but only when the flow's ID is GAS). The first of these, GASBW, will return the ATOM array, given the GAS.COMP substructure; the second, GASNM, returns the total number of moles per kilogram of gas flow, given the GAS.COMP substructure. The gas properties codes make use of the ATOM array, which should be specified before the gas properties code is called. Those models that adjust the molar fractions of a gas flow (e.g., mixers, combustors, and fuel cells) should call GASBW with the new GAS.COMP to obtain GAS.ATOM before any calls are made to GP.

The gas properties codes offer another feature that is sometimes useful in systems modeling -- the ability to do frozen chemistry calculations. When the externally declared variable GASFRZ is set to 0 or 1, the equilibrium calculations are turned on or off. Using this feature, processes that are not at chemical equilibrium can be handled, at least approximately. (Changes in the molar fractions of the gas as the temperature and pressure are changed must be handled within the models or by some other means when GASFRZ is set to 1.) The calculation of the thermodynamic properties proceeds as before, based on the specified species molar fractions. The GASFRZ variable may be turned on and off selectively (using the PLI statement within the SALT input) at various places within the system analysis.

The GP code has an externally declared variable (GPPRT) that can be set to 0 or 1 to print out the values of the model name, flow name, temperature, pressure, enthalpy, entropy, mass flow rate, and SW after every property call. The GPPRT variable is often useful for debugging a new system configuration. Another externally declared variable (GASPRT) will provide a similar service for just the gas properties codes. In this case, GASPRT can also be set to 2 in order to obtain the actual iterations performed in the equilibrium calculations. This setting should be used with discretion; it can produce quite a lot of output.

The overall organization of the properties codes is as follows. For each ID type (H2O, THR, etc.), several procedures are available; the name of each procedure begins with the ID characters (for ID = H2O, STM is used as an alias), followed by IN, TP, HP, SP, SAT, or WK. Thus, for the ID type THR, the procedures THRIN, THRTP, THRHP, THRSP, THRSTAT, and THRWK can be found. The IN procedure is used to obtain the data from the THRDATA file; the TP, HP, and SP procedures are used to perform the calculations for the three different values of the SW parameter; SAT is used when GPSAT is called; and the WK procedure is a general work procedure that is called by the other entries. Not all of the different ID types include all the different entries. For example, LIQIN and LIQWK are the only entries available when ID is equal to LIQ; in this

case, the SW value is passed directly to LIQWK to perform the functions of the other entries. Also, some of the properties codes do not have saturation properties, because they model only a single fluid phase; this limitation applies to the JAN, LIQ, and GAS properties codes.

3 ADDITION OF NEW MODELS AND FLOW TYPES

3.1 INTF FILE

The SALT code has been designed so that new models can be added with as little effort as possible, while retaining the ability to handle models with arbitrary levels of complexity. The key to maintaining the flexibility required to handle arbitrary models is the mechanism by which the constructed driver establishes its interface with the component models. Numerous approaches have been employed in this problem, but the one ultimately used is relatively simple and has proven to be one of the most flexible solutions. The technique is simply to read in from an external file those variables that must be declared for each component model used in a given system analysis. These variables may be the names of model parameters that are passed to the component models, the names of entry variables, or the names of other variables that need only be included when a particular model is used.

The file that contains the interface information, called INTF, is used by the SALT code when constructing the PL/I driver for the problem under consideration. The form of the interface consists of a series of header statements, each with a 0, 1, 2, or 3 in column one, followed by other data (depending on the number). The type-0 statements are used to read in information that will be needed by the system models in locating specific substructures of the models; the type-1 statements define the model interfaces; the type-2 statements define the flow interfaces; and the type-3 statements define additional coding that will be unconditionally inserted into the PL/I driver.

The specific form of the type-0 statement is as follows:

```
OSUBSTRUCTURE_NAME  SUBSTRUCTURE_HEAD_PTR
```

Here, SUBSTRUCTURE_NAME is the name of the model substructure that will be included within a linked list for use in system models, and SUBSTRUCTURE_HEAD_PTR is a pointer variable that points to the beginning of this linked list.

Some of the substructures to be included in a linked list have variable names -- such as FLC, FLH, FLC1, etc. -- so the character "*" may be used at the end of a SUBSTRUCTURE_NAME to refer to all substructures that begin with the specified SUBSTRUCTURE_NAME. All substructures that begin with those common characters will then be included in the same linked list. To create a linked list for all of the flow substructures starting with the characters "FL," for instance, one would write the following:

```
OFL*      FLOW_HEAD_PTR
```

The type-1 statement header takes the form of a 1 in column one, followed by the name of the model. Statements following this header statement represent the PL/I declarations of the model structure variable, followed by the declaration of the model entry points, each on a separate line. For example, the interface statements for the ST model are as follows:

```

1ST
DCL
  1 ST BASED,
    2 NAME CHAR(16),
    2 FLC1,
      3 FNAME CHAR(16),
      3 ID CHAR(4),
      3 ATOM(8) FLOAT(16),
      3 PROP,
        4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
      3 COMP,
        4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
          XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
      3 SOL,
        4 WTF FLOAT(16),
    2 FLC2,
      3 FNAME CHAR(16),
      3 ID CHAR(4),
      3 ATOM(8) FLOAT(16),
      3 PROP,
        4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
      3 COMP,
        4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
          XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
      3 SOL,
        4 WTF FLOAT(16),
    2 PARM,
      3 DDNAME CHAR(7),
      3 MODE CHAR(15),
      3 EXIT PRES FLOAT(16),
      3 EFFICIENCY FLOAT(16),
      3 MECH EFF FLOAT(16),
      3 SR FLOAT(16),
      3 EXT MASS FLOAT(16),
      3 FLOW FACT FLOAT(16),
      3 EXHAUST LOSS FLOAT(16),
      3 DM FLOAT(16),
      3 WV FLOAT(16),
      3 WHEEL SPEED FLOAT(16),
      3 CONS FLOAT(16),
      3 VOL FLOW RATE FLOAT(16),
      3 PRINT FIXED BIN(15),
    2 POWER,
      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
    2 E LOSS,
      3 PTR POINTER,
    2 COST FLOAT(16);
DCL STC ENTRY;
DCL STOUT ENTRY;

```

Other variables may be declared within the interface for each model, but no particular variable should be declared in more than one model interface. If other variables are declared, they should follow the declaration of the model structure variable.

Although the statements following the header are simply PL/I declarations, the SALT code performs some checking of this input in order to retain the names of the 2-level substructures and the entry names. In the case of the model declarations, some programmers use the "style" of writing the comma before the next line rather than at the end of the line. The SALT code checks for the presence of the string "2" in locating the 2-level substructures, but it will not properly locate those preceded by ",2."

Usually, no executable statements appear within the interface. However, it is possible to include whole PL/I procedures. Such procedures are simply compiled right into the PL/I driver code along with the other statements in the interface. Only statements belonging to the interfaces of models actually used in the system problem will be brought into the driver code.

The type-2 statement header takes the form of a 2 in column one, followed by the name of a flow type. Statements following this header declare the form of the flow. For example, the interface statements for the STM flow are as follows:

```
2STM
DCL 1 STM BASED,
    2 NAME CHAR(16),
    2 ID CHAR(4),
    2 ATOM(8) FLOAT(16),
    2 PROP,
    3 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
    2 COMP,
    3 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    2 SOL,
    3 WTF FLOAT(16);
```

As was the case for the model interfaces, only those flow-type interfaces that are actually used in the system problem will be included in the driver code.

The type-3 statement header takes the form of a 3 in column one, followed by any comments or by blanks. The lines following this header statement, up to the occurrence of another header line, will be included in the driver code. For example, declarations of unit-conversion variables might be included.

The different types of interfaces may be freely mixed, but for readability it is useful to group all of the different types together. The INTF file may also reside in more than one data set and be concatenated together at run time.

3.2 ADDITION OF NEW MODELS

New PL/I component models may be added to the library of SALT models at any time. To do this, one need only take into account a few rules necessary for the SALT code to interface correctly with the model and add the appropriate interface statements to the INTF file.

The basic rules to be observed in developing a new model are as follows. First, a model may have any number of entry points, but the first several characters of the entry name should be the name of the model as it appears in a PROCESS statement. Additional letters (up to the PL/I limit of seven characters for entry names) may be used to define the separate entry points. Thus, the heat-exchanger model (HX) has the entry points HXH, HXC, and HXOUT. (When a model is referred to within a PROCESS statement without using the colon and entry name, the "C" entry -- as in HXC -- will be called. The main calculational entry should be specified as this "C" entry when developing new models.)

The second basic rule concerns the arguments that are passed to these entry points. The first argument to all entry points is a pointer to the model structure variable. Any additional arguments that follow are pointers to the flow variables, which should be arranged in the order of pass-through flows, input flows, and output flows to the model. For the "OUT" entry (e.g., STOUT), the only argument should be the model structure variable.

The third rule concerns the model structure variables. The 1-level name should be the name of the model. The first 2-level name should be NAME, declared as a CHAR(16) variable. This variable is always defined by the SALT code as being the name of the model (including user-defined label) as it is called within the system problem.

Other 2-level structures may be defined by the model developer to store various input and output data from the model. The 2-level name POWER already has a special structure; if POWER is used, this structure should correspond to that used by the existing models.

Beyond these basic rules concerning the naming of the entry points, the arrangement of the arguments, and the naming of the model substructure variables, any type of PL/I coding may be used in the model. A model may even call FORTRAN subroutines to perform its calculations. It is important, however, that the output from a model be a function only of the input flows and model parameters. Each time a model is called with the same input values, it should return the same output values.

Once a model has been developed and debugged, it is compiled into the model load library, and the interface statements are added to the INTF file. Usually, the interface file will consist of the model structure variable and the declaration of the entry points. The model structure will already have been written, so it needs only to be copied from the model into the interface file (with the possible addition of initial attributes to define default input values). These additions to the INTF file should take only a few minutes of editing time.

3.3 ADDITION OF NEW FLOW TYPES

New flow types (where "flow types" refers to the structure of the flow variables) to be processed by newly developed models may be added to the SALT code at any time. Additional steam, liquid, or gas flows that are structurally the same as STM, LIQ, or GAS are generated as needed within a system problem, using the labeling option for flows

(i.e., STM_1, STM_2, etc). At present, the STM, GAS, and LIQ flows are technically of the same type; they have been defined as separate flow types to furnish the user of SALT with some variety in the flow names available for use.

The addition of a new flow type is accomplished by adding the PL/I declaration of the flow variables to the INTF file. Of course, this new flow will not be usable unless new component models have been written to accommodate this new flow type. If system models exist that print out flows, these models may have to be modified to accept the new flow types. Nothing else needs to be done to add a new flow type.

As was the case for the model structures, each flow variable should have as its first 2-level element the variable NAME, declared as a CHAR(16) variable. This variable will be assigned the name of the flow as it is used in the system problem by the SALT code.

4 NUMERICAL PROCEDURES

This chapter discusses the main mathematical procedures used by SALT in terms of their use and calling arguments. Three basic procedures are used in performing a steady-state system analysis: SOV, a one-dimensional equation solver; SOLVG, a multidimensional equation solver; and OPT, a multidimensional optimizer.

4.1 PROCEDURE FOR USING SOV

The SOV procedure is used quite often, both in the models and in the properties procedures. However, it is never used by the driver code as constructed by SALT. Within the driver code, even one-dimensional problems are solved using SOLVG. The SOV procedure attempts to solve an equation using a secant method, but some safeguards are employed when a root of the equation has been bracketed.

Like all the numerical procedures employed by SALT, SOV makes use of an inverse calling sequence in which the iterative loop is not within the numerical procedure but within the calling program. This arrangement permits the numerical procedures to be called at any time to solve a new problem, even when the iterations of the previous problem are not finished (i.e., nested problems can be solved without resorting to recursive procedures). The coding necessary to solve a problem is simpler, because additional procedures that would define equation residuals are not needed.

The arguments to the SOV procedure are as follows:

X -- Independent variable of the problem.

F -- Equation residual at X.

I -- Iteration counter, which should be incremented by 1 for each iteration (starting at 1).

DEL -- Initial perturbation in the X variable.

ACC -- Termination criterion; whenever $ABS(F) < ACC$, I is set to 1000.

MAXI -- Maximum number of iterations permitted.

SOVPRT -- Print switch used to produce printed output of the iterations (if set to 1), or to produce no output except on failure to converge (if set to 0).

LABEL -- Character string used in the printout to delimit which call to SOV is being printed.

With these calling arguments, which are all inputs, the coding necessary to solve $f(X) = 0.0$ is as follows:

```
DO I=1 TO MAXI;
  F=f(X);
  CALL SOV(X,F,I,DEL,ACC,MAXI,SOVPRT,LABEL);
END;
```

Here, it is assumed that each variable (except F) has been given a value before entering the loop. Since I is set to 1000 on convergence, MAXI must always be less than 1000. At each iteration, X is given a new value on exit from SOV, until the termination or maximum number of iterations is reached.

4.2 PROCEDURE FOR USING SOLVG

The SOLVG procedure is a multidimensional nonlinear equation solver that uses a hybrid quasi-Newtonian-update/steepest-descent technique. Like SOV, the procedure uses the inverse calling mechanism. Unlike SOV, however, SOLVG attempts to find a root to the equations within some specified lower and upper bounds on X.

The arguments to SOLVG are as follows:

X -- Array of independent variables of the problem.

FSQ -- Sum of the squares of the equation residuals.

F -- Array of equation residuals.

N -- Problem dimension.

BL -- Array of lower bounds on X.

BU -- Array of upper bounds on X.

I -- Iteration counter, used as with SOV.

DEL -- Parameter used to calculate the initial perturbations in the X-array elements for calculating the Jacobian of the system of equations.

ACC -- Termination criterion; when $FSQ < ACC$, I is set to 1000.

MAXI -- Maximum iterations permitted.

SOLVGPRT -- Print switch, used to print out details of the iterations. When SOLVGPRT is zero, no output is obtained; when SOLVGPRT is 2, the maximum output is obtained.

LABEL -- Character string used to delimit printout.

All of these variables (except FSQ) are inputs and should be assigned values before SOLVG is called. The coding necessary to solve the system of equations, $f_1(X) = 0.0$, $f_2(X) = 0.0, \dots$, $f_n(X) = 0.0$, would be as follows:

```
DO I=1 TO MAXI;
  F(1)=f1(X);
  F(2)=f2(X);
  .
  .
  .
  F(N)=fn(X);
  CALL SOLVG(X,FSQ,F,N,BL,BU,I,ACC,DEL,MAXI,SOLVGPR,T,LABEL);
END;
```

4.3 PROCEDURE FOR USING OPT

OPT solves a nonlinearly constrained, nonlinear optimization problem of the following form:

```
MIN F(X)
over all X(i) i=1 TO N such that
BL(i)<X(i)<BU(i), i=1 TO N
Ci(X)=0.0, i=1 TO MEQ
Ci(X)>0.0, i=MEQ+1 TO M
```

OPT uses a sequential quadratic programming procedure with a BFGS (Broyden, Fletcher, Goldfarb, and Shanno) update to develop the Hessian of the Lagrangian function of the problem.

The calling sequence to OPT includes the following arguments:

X -- Array of independent variables of the problem.

F -- Objective function to be minimized.

C -- Array of constraints.

MEQ -- Number of constraints that are equalities.

BL -- Array of lower bounds on X.

BU -- Array of upper bounds on X.

I -- Iteration counter.

ACC -- Termination criterion; when the first Kuhn-Tucker condition is satisfied to within ACC, I is set to 1000.

DEL -- Parameter used to define the perturbations in X for calculating gradients.

MAXI -- Maximum iterations permitted.

OPTPRT -- Print switch for producing output -- 0 for no output, 4 for maximum output.

LABEL -- Character-string delimiter for the printout.

All of these arguments are inputs and should be assigned values before OPT is called. The coding necessary to solve the nonlinearly constrained optimization problem then becomes:

```
DO I=1 TO MAXI;
  F= f(X);
  C(1)=C1(X);
  C(2)=C2(X);
  .
  .
  .
  C(M)=CM(X);
  CALL OPT(X,F,C,MEQ,BL,BU,I,ACC,DEL,MAXI,OPTPRT,LABEL);
END;
```

APPENDIX A: CODE LISTING

APPENDIX A: CODE LISTING

A.1 COMBUSTOR MODEL

A.1.1 Description of Model

The CB model, representing a generic combustor, requires three flows; the first two are inputs, and the third is an output. The first flow represents the fuel input and is of the generic type FUEL. The second flow represents any oxidizing flow, while the third represents the combustion-gas output; these latter two flows are of the generic type GAS. The CB model, as a generic combustor, can also model a gasifier, where the output gas-flow conditions are at chemical equilibrium. Options are also provided for ash removal and potassium injection (used for magnetohydrodynamic systems).

The parameters of the CB model are as follows:

HEAT_LOSS_FRAC -- Specified fraction of the thermal input (based on the higher heating value of the fuel) lost from the combustor due to heat loss.

FUEL_M -- Calculated value of the fuel mass after any ash removal.

FUEL_HHV -- Corrected higher heating value of the fuel after any ash removal.

ASH_M -- Calculated amount of mass removed from the fuel as ash.

ASH_M_FUEL -- Calculated amount of mass left in the fuel as ash.

ASH_DET -- Specified weight fraction of the ash within the fuel after any ash rejection.

FUEL_HEAT_FORM -- Heat of formation of the fuel at a pressure of 1 atm and a temperature of 298.16 K.

H2O_M_FUEL -- Calculated amount of H₂O left in the fuel.

SLURRY_CONC -- Calculated weight fraction of solid fuel to total weight of fuel (useful when CB is modeling a gasifier).

CARBON_BURNOUT -- Specified fraction of the carbon in the fuel that is actually combusted; the rest of the carbon is carried over with any ash carry-over.

K_MASS -- Calculated weight of potassium in the output gas flow.

K_FRAC -- Specified weight fraction of potassium in the output gas flow (used to model potassium seed injection in MHD systems).

OX_M -- Calculated mass of oxygen needed for stoichiometric combustion of the fuel.

STOICH -- Calculated fraction of the mass of total oxygen in oxidizer flow to the mass, OX_M.

PRES_DROP_FRAC -- Specified fraction of the input oxidizer pressure representing the pressure drop through the combustor.

BC(8), BO(8), and BG(8) -- Calculated elemental mass fractions for the fuel, oxidizer, and output gas flows, respectively.

A.1.2 Declaration Structure

```
* PROCESS NAME('CBC');
CBC: PROC(CB_P, FUEL_P, AIR_P, GAS_P);

DCL (CB_P, FUEL_P, AIR_P, GAS_P) POINTER;
DCL 1 CB BASED(CB_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 HEAT_LOSS_FRAC FLOAT(16),
    3 FUEL_M FLOAT(16),
    3 FUEL_HHV FLOAT(16),
    3 FUEL_HEAT_FORM FLOAT(16),
```

```

3 ASH_M FLOAT(16),
3 ASH_M_FUEL FLOAT(16),
3 ASH_DET FLOAT(16),
3 H2O_M_FUEL FLOAT(16),
3 SLURRY_CONC FLOAT(16),
3 CARBON_BURNOUT FLOAT(16),
3 K_MASS FLOAT(16),
3 OX_M FLOAT(16),
3 STOICH FLOAT(16),
3 PRES_DROP_FRAC FLOAT(16),
3 K_FRAC FLOAT(16),
3 BC(8) FLOAT(16),
3 BO(8) FLOAT(16),
3 BG(8) FLOAT(16),
2 POWER,
3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
2 COST FLOAT(16);
DCL 1 FUEL BASED(FUEL_P),
2 FNAME CHAR(16),
2 PROP,
3 (TEMP,MASS,HHV) FLOAT(16),
2 WEIGHTS,
3 (C,H,O,N,S,CL,H2O,ASH) FLOAT(16);
DCL GAS BASED(GAS_P) LIKE FLC2;
DCL AIR BASED(AIR_P) LIKE FLC1;
DCL AW(8) FLOAT(16) INIT(39.948,12.01115,1.00797,39.102,
14.0067,15.9994,32.064,35.453);
DCL (GP,GASNM) ENTRY,
(FRAC,NMOLE,TOT) FLOAT(16),
FL LIKE FUEL.WEIGHTS;

```

Initialize power and set power loss to the environment.

```

POWER=0.0;
POWER.LOSS = HEAT_LOSS_FRAC*FUEL.MASS*FUEL.HHV;

```

Set pressure drop through the combustor.

```

AIR.PRES = AIR.PRES*(1.0 - PRES_DROP_FRAC);

```

Using the specified ASH_DET, calculate the fuel's weight fractions and its higher heating value after any ash removal.

```

FRAC=(1.0-ASH_DET)/(1.0-FL.ASH);
FUEL_M=FUEL.MASS/FRAC;
FUEL_HHV=FRAC*FUEL.HHV;
FL=FL*FRAC;
FL.ASH=ASH_DET;

```

Calculate the ash mass removed by the ash-removal process and the amount of ash and water remaining in the fuel.

```

ASH_M=FUEL.MASS-FUEL_M;
ASH_M_FUEL=FL.ASH*FUEL_M;
H2O_M_FUEL=FL.H2O*FUEL_M;

```

Calculate the slurry concentration (for use in modeling gasifiers).

```

SLURRY_CONC=(FUEL_M-H2O_M_FUEL)/FUEL_M;

```

Using the fuel weight fractions, calculate the weights of the individual elements found in 1 kg of fuel.

```

BC(1)=0.0;
BC(2)=FL.C*CARBON_BURNOUT;
BC(3)=FL.H+0.111902*FL.H2O;
BC(4)=0.0;
BC(5)=FL.N;
BC(6)=FL.O+0.888098*FL.H2O;
BC(7)=FL.S;
BC(8)=FL.CL;

```

Assuming 100% oxidation of the carbon, sulfur, and hydrogen within the fuel, calculate the amount of oxygen required.

```

OX_M=2.6641*FL.C*FUEL_M+7.93645*FL.H*FUEL_M+0.99797*FL.S*FUEL_M-
FL.O*FUEL_M;

```

Calculate the total weights of the elements within the gas produced when burned at stoichiometric conditions. (This calculation will be used to determine the heat of formation of the fuel.)

```

BG=BC*FUEL_M;
BG(6)=BG(6)+OX_M;
BG=BG/(FUEL_M+OX_M);

```

Set the conditions of the gas burned at stoichiometric conditions and then call the properties code to determine the enthalpy of the gas. This gas enthalpy will then be used to determine the heat of formation of the fuel.

```

GAS.TEMP=298.16;
GAS.PRES=1.0;
GAS.ATOM=BG/AW;
GAS.ID='GAS';
GAS.WTF=(ASH_M_FUEL+FL.C*(1.0-CARBON_BURNOUT))/(FUEL_M+OX_M);
CALL GP(NAME,GAS,1B);

```

Calculate the amount of energy needed to vaporize the water within the gas.

```

GASNM(GAS.COMP,NMOLE);
FRAC=1.05065E3*2.324444E3*18.01534*NMOLE*GAS.XH2O;

```

Using the calculated gas enthalpy, the amount of water vaporization energy, and the higher heating value of the fuel, calculate the heat of formation of the fuel.

```

FUEL_HEAT_FORM=((FUEL_M+OX_M)*(GAS.ENTH-FRAC)+
FUEL_M*FUEL_HHV)/FUEL_M;

```

Calculate the weight fraction of any potassium that may be injected into the gas (for use in open-cycle MHD systems).

```

K_MASS=(FUEL_M+OX_M)*K_FRAC/(1.0-K_FRAC);

```

Calculate the gas mass that will leave the combustor.

```

GAS.MASS=FUEL_M+AIR.MASS+K_MASS;
GAS.WTF=ASH_M_FUEL/GAS.MASS;

```

Using the kg-atoms/kg fractions of the elements within the oxidizer stream, plus the fuel element fractions and any added potassium, calculate the kg-atoms/kg fractions of the elements in the combustion gas.

```

STOICH=BO(6)*AIR.MASS/OX_M;
BG=BO*AIR.MASS+BC*FUEL_M;
BG(4)=BG(4)+K_MASS;
BG=BG/GAS.MASS;

```

Set the exit conditions of the combustion gases and call the properties code to determine the other state values for the gas (i.e., density, entropy, etc.)

```

GAS.ATOM=BG/AW;
GAS.VEL=FLC2.VEL;
GAS.PRES=AIR.PRES;
GAS.ENTH=(AIR.ENTH*AIR.MASS+FUEL_M*FUEL_HEAT_FORM-POWER.LOSS)
/GAS.MASS;
GAS.TEMP=FLC2.TEMP;
CALL GP(NAME,GAS,10B);

```

Save the input air flow and the exiting combustion-gas flow.

```

FLC1=AIR;
FLC2=GAS;
RETURN;

```

```

CBOUT: ENTRY(CB P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT(
'FUEL MASS BURNED = ',PARM.FUEL_M,
'FUEL HHV = ',PARM.FUEL_HHV,
'FUEL HEAT OF FORM AS BURNED = ',PARM.FUEL_HEAT_FORM,
'HEAT LOSS FRACTION = ',PARM.HEAT_LOSS_FRAC,
'STOICHIOMETRY = ',PARM.STOICH,
'CARBON BURNOUT = ',PARM.CARBON_BURNOUT,
'ASH MASS REMOVED = ',PARM.ASH_M,
'ASH MASS IN FUEL = ',PARM.ASH_M_FUEL,
'WATER MASS IN FUEL = ',PARM.H2O_M_FUEL,
'SLURRY CONCENTRATION = ',PARM.SLURRY_CONC,

```

```

'POTASSIUM MASS = ',PARM.K MASS,
'SEED FRACTION = ',PARM.K_FRAC)(COL(10),A,E(12,5));
PUT SKIP(2) EDIT('FUEL ELEMENT FRACTIONS (AS BURNED)')(COL(10),A)
('ARGON','CARBON','HYDROGEN','POTASSIUM','NITROGEN','OXYGEN',
'SULFUR','CHLORINE',PARM.BC)(COL(15),8 A(10),COL(13),8 F(10,6));
PUT SKIP(2) EDIT('OXIDIZER ELEMENT FRACTIONS')(COL(10),A)
('ARGON','CARBON','HYDROGEN','POTASSIUM','NITROGEN','OXYGEN',
'SULFUR','CHLORINE',PARM.BO)(COL(15),8 A(10),COL(13),8 F(10,6));
PUT SKIP(2) EDIT('GAS ELEMENT FRACTIONS')(COL(10),A)
('ARGON','CARBON','HYDROGEN','POTASSIUM','NITROGEN','OXYGEN',
'SULFUR','CHLORINE',PARM.BG)(COL(15),8 A(10),COL(13),8 F(10,6));
END CBC;

```

A.2 COMPRESSOR MODEL

A.2.1 Description of Model

The compressor model (CP) requires one pass-through flow of the generic type of GAS. A simplified off-design option is also provided. In the design mode, the model obtains the exit flow conditions by calculating an isentropic compression to a specified exit pressure and then corrects for a specified isentropic efficiency. In off-design use, a nondimensional mass factor is also calculated.

In the off-design mode, the model requires an initializing call to CPIN to obtain a table of pressure ratios vs. the mass factor (normalized by the design-point mass factor). During flow processing, the model then uses this table to calculate (based on the inlet mass factor) the pressure ratio and, hence, the exit pressure. The model then proceeds as in the design mode.

The parameters of the CP model are as follows:

DDNAME -- Character string representing the file name of the off-design pressure-ratio-vs.-mass-factor table. This variable is not needed in the design mode. The information in this file consists of (1) an integer specifying the number of pressure-ratio values, followed by (2) the list of pressure-ratio values and by (3) the list of normalized mass-factor values.

MODE -- Character string representing either DESIGN mode or OFF-DESIGN mode.

EXIT_PRES -- Specified exit pressure for the design mode.

EFFICIENCY -- Specified isentropic efficiency of the compression process.

MASS_FACT -- Calculated mass factor for use in off-design calculations. This factor is an output in the design mode, but it must be an input in the off-design mode.

M_FACT -- Calculated value of the normalized mass factor used in off-design calculations. This variable is assigned the value 1 at the design point.

PRES_RATIO -- Calculated pressure ratio across the compressor.

A.2.2 Declaration Structure

```
* PROCESS NAME('CPC');
CPC: PROC( CP_P , AIR_P );

DCL (CP_P, AIR_P) POINTER;
DCL 1 CP BASED(CP_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 DDNAME CHAR(7),
    3 MODE CHAR(10),
    3 EXIT_PRES FLOAT(16),
    3 EFFICIENCY FLOAT(16),
    3 MASS_FACT FLOAT(16),
    3 M_FACT FLOAT(16),
    3 PRES_RATIO FLOAT(16),
    2 POWER,
    3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
    2 PRATIO,
    3 PTR POINTER,
    2 COST FLOAT(16);
DCL AIR BASED(AIR_P) LIKE FLC;
DCL (GP, TABC, TABIN) ENTRY,
    SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
      FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*)),
    (COV INIT(101325.0)) FLOAT(16);
```

Initialize power to zero.

```
POWER = 0.0E0;
```

Save the inlet values of the input flow.

```

FLC=AIR;
IF MODE='DESIGN' THEN
DO;

```

If in the design mode, call the procedure CP1, which will perform the calculations of the compression process.

```

CALL CP1(EXIT_PRES,EFFICIENCY);

```

Calculate the value of the mass factor to be used in any off-design run.

```

MASS_FACT=FLC.MASS/SQRT(FLC.PRES*COV*FLC.RHO);
M_FACT=1.0;

```

Calculate the pressure ratio across the compressor.

```

PRES_RATIO=EXIT_PRES/FLC.PRES;
END;
ELSE
DO;

```

If in the off-design mode, use the mass factor from the design-mode run to calculate the normalized inlet mass factor for the off-design conditions.

```

M_FACT=FLC.MASS/SQRT(FLC.PRES*COV*FLC.RHO)/MASS_FACT;

```

Using the normalized mass factor, call the response surface (i.e., the pressure ratio vs. normalized mass factor) to obtain the pressure ratio.

```

TABC(PRATIO,M_FACT,PRES_RATIO);

```

Calculate the exit pressure.

```

EXIT_PRES=PRES_RATIO*FLC.PRES;

```

Calculate the conditions at the exit of the compression process by calling CP1.

```

CALL CP1(EXIT_PRES,EFFICIENCY);
END;

```

Calculate the power consumed during the compression.

```

POWER.CONSUMED=FLC.MASS*(AIR.ENTH-FLC.ENTH);
COST = 0.0;

```

Save the exit flow.

```

    FLC = AIR;
    RETURN;
    CP1: PROC(P_OUT,EFF);
    DCL (P_OUT,EFF) FLOAT(16);

```

Set the value of the exit pressure.

```
AIR.PRES=P_OUT;
```

Calculate the conditions of the flow at the given exit pressure and with the inlet value of the entropy (i.e., at isenthalpic conditions).

```
CALL GP(NAME,AIR,11B);
```

Calculate the enthalpy at the exit of the compression using the isenthalpic enthalpy and the efficiency of the process.

```
AIR.ENTH=FLC.ENTH+(AIR.ENTH-FLC.ENTH)/EFF;
```

Call the properties code to obtain the other state variables of the flow at the exit of the compressor.

```

    CALL GP(NAME,AIR,10B);
    END CP1;

    CPIN: ENTRY(CP_P);
    IF MODE='DESIGN' THEN
        CALL TABIN(PRATIO,DDNAME);
    RETURN;

    CPOUT: ENTRY(CP_P);
    PUT SKIP EDIT(' ',NAME)(COL(4),A);
    PUT EDIT('MODE = ',MODE,
    'EXIT PRES = ',EXIT PRES,
    'EFFICIENCY = ',EFFICIENCY,
    'MASS FACTOR = ',MASS_FACT,
    'M FACTOR = ',M_FACT,
    'PRESSURE RATIO = ',PRES_RATIO)
    (SKIP(2),COL(10),A,A,6 (SKIP,COL(10),A,E(13,5)));
    END CPC;

```

A.3 DEAERATOR MODEL

A.3.1 Description of Model

The deaerator model (DEAR) requires two steam flows, the first of which is a pass-through flow representing not only one of the input flows, but also the output flow from the model. The DEAR model is also a demand-type model, requiring that the exit flow be saturated.

The only parameter of the DEAR model is QUAL, the output flow quality from the model. For proper modeling of a deaerator, this parameter should be constrained to equal zero by imposing some system constraint.

A.3.2 Declaration Structure

```
* PROCESS NAME('DEARC');
DEARC: PROC(DEAR_P, F1_P, F2_P);

DCL (DEAR_P, F1_P, F2_P) POINTER;
DCL 1 DEAR BASED(DEAR_P),
  2 NAME CHAR(16),
  2 FLC1,
  3 FNAME CHAR(16),
  3 ID CHAR(16) VARYING,
  3 ATOM(8) FLOAT(16),
  3 PROP,
  4 (T,P,H,S,Q,R,V,M) FLOAT(16),
  3 COMP,
  4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
  3 SOL,
  4 WTF FLOAT(16),
  2 FLC2,
  3 FNAME CHAR(16),
  3 ID CHAR(16) VARYING,
  3 ATOM(8) FLOAT(16),
  3 PROP,
  4 (T,P,H,S,Q,R,V,M) FLOAT(16),
  3 COMP,
  4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
  3 SOL,
  4 WTF FLOAT(16),
  2 PARM,
  3 QUAL FLOAT(16);
DCL GP ENTRY,
  (KE,PC,MASS,R,S,Q,HL,HS) FLOAT(16);
DCL F1 BASED(F1_P) LIKE FLC1;
DCL F2 BASED(F2_P) LIKE FLC2;
```

Save the inlet flow.

```
FLC2=F2;
```

Add the two inlet flow masses together.

```
MASS=F1.M+F2.M;
```

Calculate the total kinetic energy of the inlet flows.

```
KE=F1.M*F1.V**2+F2.M*F2.V**2;
```

Set the exit pressure from the deaerator as the minimum of the inlet pressures.

```
F1.P=MIN(F1.P,F2.P);
```

Set the exit velocity as that which will give the same total kinetic energy as the combined inlet flows.

```
F1.V=SQRT(KE/MASS);
```

Set the exit enthalpy to equal the sum of the inlets.

```
F1.H=(F1.M*F1.H+F2.M*F2.H)/MASS;  
F1.M=MASS;
```

Call the properties code to determine the other state variables of the flow.

```
CALL GP(NAME,F1,10B);
```

Save the value of the exit flow quality.

```
PARM.QUAL=F1.PROP.Q;
```

Save the value of the exit flow in FLC1.

```
FLC1=F1;  
RETURN;
```

```
DEAROUT: ENTRY(DEAR_P);  
  PUT SKIP EDIT(' ',NAME)(COL(4),A)  
    ('QUAL=',PARM.QUAL)(SKIP,COL(10),A,E(12,4));  
END DEARC;
```

A.4 GAS-DIFFUSER MODEL

A.4.1 Description of Model

The gas-diffuser model (DF) requires one pass-through flow of the generic type GAS. The parameters of the DF model are as follows:

EXIT_VEL -- Specified exit velocity of the gas flow.

PRES_RECOVERY_COEF -- Specified value of the pressure-recovery coefficient, defined as the actual pressure drop across the diffuser divided by the difference in the total static pressure at the diffuser inlet.

PRINT -- Specified print switch; if set to a number greater than zero, this switch will print out the iterations within the model used in calculating the total inlet pressure.

A.4.2 Declaration Structure

```

* PROCESS NAME('DFC');
DFC: PROC(DF_P, GAS_P);

  DCL (DF_P, GAS_P) POINTER;
  DCL 1 DF BASED(DF_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 EXIT_VEL FLOAT(16),
    3 PRES_RECOVERY_COEF FLOAT(16),
    3 PRINT FIXED BIN(15),
    2 COST,
    3 TOTAL FLOAT(16);
  DCL GAS BASED(GAS_P) LIKE FLC,
    (Q, R, S, DIFFERENCE) FLOAT(16),
    I FIXED BIN(15),
    SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
      ,FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*)),
  GP ENTRY;

```

Save the inlet gas flow and then temporarily set the total enthalpy in FLC.ENTH.

```

FLC = GAS;
FLC.ENTH=GAS.ENTH+0.5*GAS.VEL**2;

```

Iterate over the gas pressure until the gas enthalpy is equal to this total enthalpy, thus obtaining the total pressure.

```

DO I=1 TO 15;
  CALL GP(NAME,GAS,11B);
  DIFFERENCE=FLC.ENTH-GAS.ENTH;
  CALL SOV(GAS.PRES,DIFFERENCE,I,1.0,20.0,15,PRINT,'DF'); END;

```

Set the flow's exit velocity to the specified value and then calculate the flow's static enthalpy.

```

GAS.VEL=EXIT_VEL;
GAS.ENTH=GAS.ENTH-0.5*GAS.VEL**2;

```

Using the pressure-recovery factor, calculate the exit pressure from the diffuser.

```

GAS.PRES=FLC.PRES+(GAS.PRES-FLC.PRES)*PRES_RECOVERY_COEF;

```

Call the properties code to determine the other state variables of the flow.

```
CALL GP(NAME,GAS,10B);
COST.TOTAL = 0.0;
```

Save the exit flow from the diffuser.

```
FLC = GAS;
RETURN;

DFOUT: ENTRY(DF P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT(
  'PRESSURE RECOVERY COEFFICIENT = ',PRES_RECOVERY_COEF,
  'EXIT VELOCITY = ',EXIT_VEL)
  (COL(10),A,E(12,5));
END DFC;
```

A.5 FUEL-DRYER MODEL

A.5.1 Description of Model

The fuel-dryer model (DRY) has two flow-processing entry points, DRYC and DRYH. The DRYC entry, which processes the fuel input and requires a pass-through flow of the generic type FUEL, performs the calculations involved in drying the fuel to a specified water fraction and calculates the heat energy required to vaporize the removed water. This entry must be called before the DRYH entry, which processes the hot-gas drying flow and should be of the generic type GAS.

The parameters of the DRY model are as follows:

H2O_DET -- Specified weight fractions of water in the dried fuel.

H2O_M -- Calculated mass of water removed from the fuel.

HEAT_REQUIRED -- Calculated energy required to vaporize the water mass removed.

A.5.2 Declaration Structure

```
* PROCESS NAME('DRYC');
DRYC: PROC(DRY_P, FUEL_P);

DCL (DRY_P, FUEL_P, FLOW_P) POINTER;
DCL 1 DRY_BASED(DRY_P),
    2 NAME CHAR(16), 2 FLH,
    3 FNAME CHAR(16),
```

```

3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 FU,
3 FNAME CHAR(16),
3 PROP,
4 (TEMP,MASS,HHV) FLOAT(16),
3 WEIGHTS,
4 (C,H,O,N,S,CL,H2O,ASH) FLOAT(16),
2 PARM,
3 HEAT_REQUIRED FLOAT(16),
3 H2O_DET FLOAT(16),
3 H2O_M FLOAT(16);
DCL FUEL BASED(FUEL_P) LIKE FU;
DCL FLOW BASED(FLOW_P) LIKE FLH;
DCL (NM,FRAC,FUEL_M) FLOAT(16);
DCL (GP,GASNM,GASBW) ENTRY;

```

Based on the specified weight fraction of the water on output (H2O_DET), calculate the new weight of fuel after drying.

```

FRAC=(1.0-H2O_DET)/(1.0-FUEL.WEIGHTS.H2O);
FUEL_M=FUEL.MASS/FRAC;

```

Adjust the fuel's higher heating value to reflect the loss of water.

```

FUEL.HHV=FRAC*FUEL.HHV;

```

Adjust the fuel's weight fractions for the new water content.

```

FUEL.WEIGHTS=FRAC*FUEL.WEIGHTS;
FUEL.WEIGHTS.H2O=H2O_DET;

```

Calculate the amount of water removed from the fuel and the amount of heat required to vaporize that amount of water. The fuel is assumed to be at atmospheric conditions.

```

H2O_M=FUEL.MASS-FUEL_M;
HEAT_REQUIRED=1050.65*2344.444*H2O_M;
FUEL.MASS=FUEL_M;
FU=FUEL;
RETURN;

```

```

DRYH: ENTRY(DRY_P, FLOW_P);

```

Calculate the molar flow rates of the drying gas species and temporarily store these rates within the COMP structure.

```
CALL GASNM(FLOW.COMP,NM);
FLOW.COMP=FLOW.COMP*NM*FLOW.MASS*(1.0-FLOW.WTF);
```

Add the number of moles of water removed from the fuel to the number of moles of water already within the drying gas.

```
FLOW.COMP.XH2O=FLOW.COMP.XH2O+H2O_M/18.01534;
```

Adjust the mass of gas to reflect the added weight of the water. Also adjust the entrained-solids weight fraction.

```
FLOW.WTF=FLOW.WTF*FLOW.MASS/(FLOW.MASS+H2O_M);
FLOW.MASS=FLOW.MASS+H2O_M;
```

Recalculate the elemental fractions within the gas flow and adjust the enthalpy of the gas to reflect the heat lost while drying the fuel.

```
CALL GASBW(FLOW.COMP,FLOW.ATOM);
FLOW.ENTH=FLOW.ENTH-HEAT_REQUIRED/FLOW.MASS;
```

Calculate the exit flow conditions of the gas by calling the properties code with the new enthalpy value as input.

```
CALL GP(NAME,FLOW,10B);
```

Save the exit gas flow.

```
FLH=FLOW;
RETURN;

DRYOUT: ENTRY(DRY_P);
PUT SKIP EDIT(' T,NAME)(COL(4),A)
('FUEL HHV=',FU.HHV,'FUEL MASS=',FU.MASS,'H2O_DET=',H2O_DET,
'H2O REMOVED=',H2O_M,'HEAT_REQUIRED=',HEAT_REQUIRED)
(COL(10),A,E(12,5));
PUT SKIP(2) EDIT('FUEL WEIGHT FRACTIONS')(COL(10),A)
('CARBON','HYDROGEN','OXYGEN','NITROGEN','SULFUR','CHLORINE',
'WATER','ASH',FU.WEIGHTS)(COL(15),8 A(10),COL(13),8 F(10,6));
END DRYC;
```

A.6 FEEDWATER-HEATER MODEL

A.6.1 Description of Model

The closed feedwater heater modeled by FH incorporates desuperheating, condensing, and drain-cooling zones. The model is set up to process the hot flows -- extracted from the turbine and from higher-pressure feedwater heaters -- in one entry and the cold feedwater flow in another. The hot-flow entry (FHH) requires one pass-through flow (representing the turbine extraction flow on input and the drain-cooler exit

flow on output) and one input flow (representing any cascaded flow from a higher-pressure feedwater heater). This hot-flow entry must be called before the cold-flow entry (FHC), which requires one pass-through flow. All of the flows used within the FH model are of the generic type STM.

The parameters of the FH model are as follows:

SUBCOOL -- Specified amount of subcooling of the drain-cooler exit flow.

HEAT -- Calculated total amount of heat transferred from the hot flows to the cold flow.

AREA -- Calculated total surface area of the desuperheating and condensing regions of the feedwater heater.

TTD -- Calculated terminal temperature difference, defined as the difference in temperature between the hot-flow exit temperature from the condensing region and the cold-flow exit temperature from the desuperheating region.

TSAT -- Calculated hot-flow saturation temperature at the pressure within the condensing region.

FW_VEL -- Specified velocity of the cold feedwater flow through the condensing region.

DCTD -- Calculated drain-cooler temperature difference, defined as the temperature difference between the hot-flow exit temperature from the heater and the cold-flow entrance temperature.

HDP -- Specified flow pressure-drop fraction. (The pressure drop is equal to this parameter times the input pressure.)

CDP(3) -- Specified array of cold-flow pressure-drop fractions through the desuperheating section, CDP(1); the condensing section, CDP(2); and the drain-cooler section, CDP(3).

A(3) -- Calculated array of heat-transfer-surface areas for the individual feedwater-heater regions: desuperheater, 1; condenser, 2; and drain cooler, 3.

Q(3) -- Calculated array of heat-transfer values for the three regions of the heater.

U(3) -- Calculated array of heat-transfer coefficients for the three regions of the heater.

LDTD(3) -- Calculated array of log mean temperature differences for the three regions of the heater.

HTEMP(4) -- Calculated end-point temperatures of the hot flow between the three regions of the heater, where HTEMP(1) is the inlet temperature and HTEMP(4) is the exit temperature from the heater. Because HTEMP(2) and HTEMP(3) represent the hot-flow condensing-region temperatures, these two temperatures are both equal to the saturation temperature.

CTEMP(4) -- Calculated cold-flow temperatures between the three regions of the heater, where CTEMP(1) is the exit temperature and CTEMP(4) is the cold-flow inlet temperature.

A.6.2 Declaration Structure

```
* PROCESS NAME('FHH');
FHH: PROC(FH_P, STM_P, STME_P);

DCL (FH_P, STM_P, STME_P, STMF_P) POINTER;
DCL 1 FH BASED(FH_P),
    2 NAME CHAR(16),
    2 FLH1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLH2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
```

```

3 COMP,
  4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
  4 WTF FLOAT(16),
2 PARM,
  3 SUBCOOL FLOAT(16),
  3 HEAT FLOAT(16),
  3 AREA FLOAT(16),
  3 TTD FLOAT(16),
  3 TSAT FLOAT(16),
  3 FW VEL FLOAT(16),
  3 DCTD FLOAT(16),
  3 HDP FLOAT(16),
  3 CDP(3) FLOAT(16),
  3 A(3) FLOAT(16),
  3 Q(3) FLOAT(16),
  3 U(3) FLOAT(16),
  3 LMTD(3) FLOAT(16),
  3 HTEMP(4) FLOAT(16),
  3 CTEMP(4) FLOAT(16);
DCL STM BASED(STM_P) LIKE FLH1,
  STME BASED(STME_P) LIKE FLH2,
  STMF BASED(STMF_P) LIKE FLC;
DCL (PC,HC,HL,HS,X,Y) FLOAT(16);
DCL (GP,GPSAT) ENTRY;

```

Set the steam pressure going into the desuperheating zone after any pressure drop.

```
STM.PRES=STM.PRES*(1-HDP);
```

Calculate the steam conditions going into the desuperheating zone.

```
CALL GP(NAME,STM,10B);
```

Save the steam temperature going into the desuperheating zone.

```
HTEMP(1)=STM.TEMP;
```

Calculate the saturation enthalpies and critical pressure at the desuperheating-zone pressure.

```

CALL GPSAT(NAME,STM,PC,HL,HS);
IF STM.PRES>PC THEN
  DO;
    PUT EDIT(' INLET PRES OF ',STM.PRES,' TO FEEDWATER HEATER ',
      FH.NAME,' IS GREATER THAN THE CRITICAL PRESSURE OF ',
      PC)(SKIP(2),A,E(12,4),A,A,SKIP,A,E(12,4));
    STOP;
  END;
IF STM.ENTH>HS THEN
  DO;

```

Calculate the heat loss by the steam to reach near-saturation conditions. (Actually, 80% of the heat needed to reach saturation conditions is used, because this value is more representative of the temperature used in calculating surface areas.)

```

      Q(1)=0.8*STM.MASS*(STM.ENTH-HS);
      STM.ENTH=STM.ENTH-0.8*(STM.ENTH-HS);
    END;
  ELSE

```

If the steam is already in the two-phase region, set Q(1) to zero.

```

      Q(1)=0.0;

```

Calculate the steam conditions coming out of the desuperheating zone.

```

      CALL GP(NAME,STM,10B);

```

Store the steam temperature coming out of the desuperheating-zone.

```

      HTEMP(2)=STM.TEMP;
      IF STME.MASS>0.0 THEN
        DO;

```

If the second flow to the heater is nonzero (i.e., there exists a cascade flow from a higher-pressure heater), then mix this flow with that coming out of the desuperheating zone.

```

          STM.ENTH=STME.MASS*STME.ENTH+STM.MASS*STM.ENTH;
          STM.MASS=STME.MASS+STM.MASS;
          STM.ENTH=STM.ENTH/STM.MASS;
        END;
      IF STM.ENTH>HL THEN
        DO;

```

Set the flow conditions coming out of the condensing zone.

```

          Q(2)=STM.MASS*(STM.ENTH-HL);
          STM.ENTH=HL;
          STM.QUAL=0.0;
        END;
      ELSE
        Q(2)=0.0;

```

Calculate the other steam properties coming out of the condensing zone.

```

      CALL GP(NAME,STM,10B);

```

Store the steam temperature coming out of the condensing zone.

```

      HTEMP(3),TSAT=STM.TEMP;
      IF SUBCOOL>0.0 THEN
        DO;

```

If any subcooling was requested, set the conditions at the subcooled point.

```

    STM.TEMP=STM.TEMP-SUBCOOL;
    HC=STM.ENTH;
    CALL GP(NAME,STM,1B);
    Q(3)=STM.MASS*(HC-STM.ENTH);
  END;
ELSE
  Q(3)=0.0;

```

Store the temperature of the steam coming out of the subcooled zone.

```

  HTEMP(4)=STM.TEMP;
  FLH1=STM;
  FLH2=STME;
  RETURN;

```

```

FHC: ENTRY(FH_P,STMF_P);

```

Save the inlet feedwater temperature.

```

  CTEMP(4)=STMF.TEMP;
  IF Q(3)=0.0 THEN
    DO;

```

If the subcooling or drain-cooler zone exists, then set the conditions of the feedwater flow coming out of the zone.

```

    STMF.ENTH=STMF.ENTH+Q(3)/STMF.MASS;
    STMF.PRES=STMF.PRES*(1-CDP(3));
    CALL GP(NAME,STMF,10B);

```

Save the temperature of the feedwater coming out of the drain-cooler zone.

```

  CTEMP(3)=STMF.TEMP;

```

Calculate the log mean temperature difference for the drain-cooler zone using the saved values of the hot and cold flow temperatures. (Note: DTMEAN will return a fictitious temperature if the hot and cold temperatures cross over.)

```

  CALL DTMEAN(HTEMP(3)-CTEMP(3), HTEMP(4)-CTEMP(4), LMTD(3));

```

Calculate the heat-transfer coefficient for the drain-cooler region if U(3) is not negative; otherwise, use the input value of U(3).

```

  IF U(3)>=0.0 THEN
    DO;
      X=CTEMP(3)*1.8-460.0;
      U(3)=5.674466*(300.0+0.33*(X-150.0))*(FLH1.MASS/5.0505)**
        0.16;
    END;

```

Calculate the heat-transfer surface area of the drain-cooler region.

```

      IF U(3)*LMTD(3)=0.0 THEN
        A(3)=Q(3)/(ABS(U(3))*LMTD(3));
      ELSE
        A(3)=1E10;
      END;
    ELSE
      DO;

```

If the drain-cooler region does not exist, set the exit feedwater temperature equal to the inlet temperature and set the log mean temperature and surface area equal to zero.

```

      CTEMP(3)=STMF.TEMP;
      LMTD(3)=0.0;
      A(3)=0.0;
    END;
  IF Q(2)=0.0 THEN
    DO;

```

If the condensing region exists, then set the conditions of the feedwater flow coming out of the region.

```

      STMF.ENTH=STMF.ENTH+Q(2)/STMF.MASS;
      STMF.PRES=STMF.PRES*(1-CDP(2));
      CALL GP(NAME,STMF,10B);

```

Save the feedwater temperature coming out of the region.

```

      CTEMP(2)=STMF.TEMP;

```

Calculate the log mean temperature difference for the region.

```

      CALL DTMEAN(HTEMP(2)-CTEMP(2), HTEMP(3)-CTEMP(3), LMTD(2));

```

If $U(2) \geq 0$, then calculate the heat-transfer coefficient; otherwise, use the input value.

```

      IF U(2)>=0.0 THEN
        DO;
          X=(TSAT-0.8*LMTD(2))*1.8-460.0;
          U(2)=5.674466*(450.0+2.0*(X-100.0))*(FW_VEL/1.640)**0.47;
        END;

```

Calculate the surface area of the condensing region.

```

      IF U(2)*LMTD(2)=0.0 THEN
        A(2)=Q(2)/(ABS(U(2))*LMTD(2));
      ELSE
        A(2)=1E10;
      END;
    ELSE
      DO;

```

If the condensing region does not exist, set the saved value of the feedwater temperature coming out of the region equal to the input temperature and set the area and log mean temperature equal to zero.

```

CTEMP(2)=STMF.TEMP;
LMTD(2)=0.0;
A(2)=0.0;
END;
IF Q(1)=0.0 THEN
DO;

```

If the desuperheating region exists, set the conditions of the feedwater flow coming out of the region.

```

STMF.ENTH=STMF.ENTH+Q(1)/STMF.MASS;
STMF.PRES=STMF.PRES*(1-CDP(1));
CALL GP(NAME,STMF,10B);

```

Save the exit feedwater temperature.

```

CTEMP(1)=STMF.TEMP;

```

Calculate the log mean temperature difference.

```

CALL DTMEAN(HTEMP(1)-CTEMP(1), HTEMP(2)-CTEMP(2), LMTD(1));

```

If $U(1) \geq 0$, then calculate the heat-transfer coefficient; otherwise, use the input value.

```

IF U(1)>=0.0 THEN
DO;
X=14.7*FLH1.PRES;
Y=FLH1.MASS-FLH2.MASS;
U(1)=5.674466*(100+0.06*(X-400))*(Y/10.10)**0.4;
END;

```

Calculate the surface area of the desuperheating region.

```

IF U(1)*LMTD(1)=0.0 THEN
A(1)=Q(1)/(ABS(U(1))*LMTD(1));
ELSE
A(1)=1E10;
END;
ELSE
DO;

```

If the desuperheating region does not exist, set saved temperature, log mean temperature, and surface area.

```

CTEMP(1)=STMF.TEMP;
LMTD(1)=0.0;
A(1)=0.0;
END;

```

Save the total surface areas of the desuperheating region and condensing region.

```
AREA=A(1)+A(2);
```

Save the terminal temperature difference.

```
TTD=HTEMP(3)-CTEMP(1);
```

Save the drain-cooler temperature difference.

```
DCTD=HTEMP(4)-CTEMP(4);
```

Save the total heat transferred in the heater.

```
HEAT=Q(1)+Q(2)+Q(3);
```

Save the exit feedwater flow.

```

FLC=STMF;
RETURN;
DTMEAN: PROC(X,Y,LMTD);
DCL (X,Y,LMTD) FLOAT(16);
SELECT;
  WHEN (X>0. & Y>0. & X=Y)
    LMTD=(X-Y)/LOG(X/Y);
  WHEN (X>0. & Y>0. & X=Y)
    LMTD=X;
  WHEN (Y<=0. & X>=0. )
    LMTD=-SQRT(ABS(Y));
  WHEN (X<=0. & Y>=0.)
    LMTD=-SQRT(ABS(X));
  WHEN (X<0. & Y<0.)
    LMTD=-(X**2 + Y**2)**0.25;
END;
END DTMEAN;

FHOUT: ENTRY(FH P);
PUT SKIP EDIT(T',NAME)(COL(4),A);
PUT SKIP(2) EDIT('HEAT=',HEAT,'SUBCOOL=',SUBCOOL,
'AREA=',AREA,'TTD=',TTD,'DCTD=',DCTD,'HDP=',HDP)
(COL(10),A,E(12,5))
('CDP =' ,CDP)(COL(10),A,3 E(13,5))
('AREAS=',A)(COL(10),A,3 E(13,5))
('HEATS=',Q)(COL(10),A,3 E(13,5))
('US =' ,U)(COL(10),A,3 E(13,5))
('LMTDS=',LMTD)(COL(10),A,3 E(13,5))
('HTEMP=',HTEMP)(COL(10),A,4 E(13,5))
('CTEMP=',CTEMP)(COL(10),A,4 E(13,5));
END FHH;
```

A.7 FLASH-TANK MODEL

A.7.1 Description of Model

This model (FLSH) represents a flash tank in which the entering flow is isenthalpically expanded through a given pressure drop. The model requires two flows: The first represents the incoming fluid (on input) or the vapor phase of the flash (on output); the second flow (an output flow) represents the liquid phase of the flash.

The parameters of the FLSH model are as follows:

PRES_DROP -- Specified pressure drop through the device.

QUAL -- Calculated quality of the input flow.

A.7.2 Declaration Structure

```
* PROCESS NAME('FLSHC');
FLSHC: PROC( FLSH_P, STM1_P, STM2_P);

DCL (FLSH_P, STM1_P, STM2_P) POINTER;
DCL 1 FLSH BASED(FLSH_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 PRES_DROP FLOAT(16),
    3 QUAL FLOAT(16);
DCL STM1 BASED(STM1_P) LIKE FLC1;
DCL STM2 BASED(STM2_P) LIKE FLC2;
```

```
DCL (ENTH_ST, ENTH_LIQ, Q, PC) FLOAT(16),
    TNAME CHAR(16),
    (GPSAT,GP) ENTRY;
```

Save the inlet flow in FLC1.

```
FLC1=STM1;
```

Initialize the value of the second (liquid) flow from the flash model as that of the inlet flow. In order to avoid losing the name of the flow, as used within the SALT inputs, it is temporarily stored in TNAME.

```
TNAME=STM2.FNAME;
STM2=STM1;
STM2.FNAME=TNAME;
```

Set the exit pressures using the specified pressure drop.

```
STM1.PRES,STM2.PRES=STM1.PRES-PRES_DROP;
```

Calculate the exit saturation enthalpies.

```
CALL GPSAT(NAME,STM1,PC,ENTH_LIQ,ENTH_ST);
IF STM1.PRES<PC THEN
  DO;
```

If the exit flow pressure is subcritical, then evaluate the quality of that flow, assuming an isenthalpic flash.

```
PARM.QUAL=(STM1.ENTH-ENTH_LIQ)/(ENTH_ST-ENTH_LIQ);
```

If the flow is subcooled or superheated, generate a fictitious quality to keep both exit flows nonzero. This requires that the exit quality of the flash flow be within the two-phase region for the model to work properly.

```
Q=MAX(MIN(0.999,PARM.QUAL),0.001);
```

Based on the quality at the exit, split the flow into the vapor and liquid flows.

```
STM1.MASS=Q*STM1.MASS;
STM2.MASS=FLC1.MASS-STM1.MASS;
```

Set the two flow enthalpies as the saturation values and call the properties code to determine the other flow conditions.

```
STM1.ENTH=ENTH_ST;
STM2.ENTH=ENTH_LIQ;
CALL GP(NAME,STM2,10B);
CALL GP(NAME,STM1,10B);
FLC1=STM1;
FLC2=STM2;
END;
```

```
ELSE
DO;
```

If the exit pressure is supercritical, stop.

```
      PUT EDIT(FLSH.NAME, ' IS BEING OPERATED ABOVE THE CRITICAL ',
        ' PRESSURE OF ',PC)(SKIP,COL(2),A,A,A,E(12,5));
      STOP;
    END;
  RETURN;

FLSHOUT: ENTRY(FLSH_P);
  PUT SKIP EDIT(' ',NAME)(COL(4),A);
  PUT SKIP(2) EDIT('QUALITY = ',PARM.QUAL,'PRES DROP = ',PRES_DROP)
    (COL(10),A,E(12,5));
END FLSHC;
```

A.8 GAS-TURBINE MODEL

A.8.1 Description of Model

The gas-turbine model (GT) requires one pass-through flow of the generic type GAS. A simplified off-design mode is also provided.

In the design mode, the exit flow conditions are calculated by means of an isentropic expansion to the specified exit pressure. The exit enthalpy is adjusted using the specified isentropic efficiency. A nondimensional mass factor is then calculated for use in off-design calculations.

In the off-design mode, an initial call to GTIN must be made to obtain a table of pressure ratios vs. normalized mass factors. This table is used during flow processing to obtain the pressure ratio for the calculated normalized mass factor. The exit flow conditions are then calculated by an expansion through this pressure ratio with the specified isentropic efficiency.

The parameters of the GT model are as follows:

DDNAME -- Specified character string representing the name of the file that contains the off-design pressure-ratio table (needed only in the off-design mode). Information contained in this file is in the following order: (1) an integer representing the number of pressure-ratio values, (2) the list of pressure ratios, and (3) the list of normalized mass factors.

MODE -- Specified character string taking the values of "DESIGN" or "OFF-DESIGN."

EFFICIENCY -- Specified isentropic efficiency of the turbine expansion.

MECH_EFF -- Specified mechanical efficiency; any thermal energy extracted through the turbine expansion is multiplied by this efficiency to obtain the useful mechanical-power output.

EXIT_PRES -- Specified design-point exit pressure (an output from the model in the off-design mode).

MASS_FACT -- Nondimensional mass factor calculated in the design mode and input in the off-design mode.

M_FACT -- Calculated nondimensional mass factor normalized by dividing by the design-point mass factor.

PRDES -- Design-point pressure ratio calculated in the design mode and input in the off-design mode.

PRES_RATIO -- Calculated pressure ratio across the turbine.

INIT -- A "flag" used by the code to initiate a reevaluation of the design-point mass factor on first entry to the model during the off-design mode. The design-point pressure ratio may not be at the maximum pressure ratio specified in the off-design pressure-ratio table; therefore, the design-point mass factor is adjusted to reflect what it would be at the maximum ratio in the table.

A.8.2 Declaration Structure

```
* PROCESS NAME('GTC');
GTC: PROC(GT_P, GAS_P);

DCL (GT_P, GAS_P) POINTER;
DCL 1 GT_BASED(GT_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 DDNAME CHAR(7),
```

```

3 MODE CHAR(10),
3 EFFICIENCY FLOAT(16),
3 MECH_EFF FLOAT(16),
3 EXIT_PRES FLOAT(16),
3 MASS_FACT FLOAT(16),
3 M_FACT FLOAT(16),
3 PRDES FLOAT(16),
3 PRES_RATIO FLOAT(16),
3 INIT_BIT(1),
2 POWER,
3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
2 PRATIO,
3 PTR POINTER,
2 COST FLOAT(16);
DCL GAS BASED(GAS_P) LIKE FLC;
DCL (COV INIT(101325.0)) FLOAT(16),
SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*)),
(CP, TABC, TABIN) ENTRY;

```

Initialize power to zero and save the inlet flow.

```

POWER = 0.0E0;
FLC = GAS;
IF MODE='DESIGN' THEN
DO;

```

If the model is being run in the design mode, call the procedure GT1 to perform the calculations of the expansion process.

```

CALL GT1(EXIT_PRES,EFFICIENCY);

```

Calculate the mass factor to be used in any off-design runs.

```

MASS_FACT=FLC.MASS/SQRT(FLC.PRES*COV*FLC.RHO);

```

Set the value of M_FACT and the pressure ratio across the turbine. (M_FACT is really only used in the off-design mode; however, it should be initialized, because it is always printed out with the other model parameters.)

```

M_FACT=1.0;
PRES_RATIO,PRDES=FLC.PRES/EXIT_PRES;
END;
ELSE
DO;

```

If the model is in the off-design mode, the exit pressure will be determined from the response surface data. However, the turbine may not have been run, during the design-mode run, at the design point of the response surface. Thus, the mass factor that would be printed out during the design-mode run would not correspond to that which would have been calculated if the design run had been made at the design-point pressure ratio of the

response surface. This mass factor is initially adjusted so that the M_FACT (from the response surface) calculated at conditions similar to the design run will give the pressure ratio of the design run.

```

IF INIT THEN
DO;
  M_FACT=1.0;
  DO I=1 TO 10;
    CALL TABC(PRATIO,M_FACT,PRES_RATIO);
    CALL SOV(M_FACT,PRES_RATIO-PRDES,I,-0.1,1E-5,10,0,'GT');
  END;
  MASS_FACT=MASS_FACT/M_FACT;
  INIT='0'B;
END;

```

Calculate the normalized mass factor to be used with the response surface data.

```
M_FACT=FLC.MASS/SQRT(FLC.PRES*COV*FLC.RHO)/MASS_FACT;
```

Call the response surface data to obtain the pressure ratio.

```
CALL TABC(PRATIO,M_FACT,PRES_RATIO);
```

Using the pressure ratio, set the exit pressure and then call GT1 to perform the actual expansion process.

```

EXIT_PRES=FLC.PRES/PRES_RATIO;
CALL GT1(EXIT_PRES,EFFICIENCY);
END;

```

Calculate the power produced.

```

POWER_PRODUCED=MECH_EFF*(FLC.ENTH-GAS.ENTH)*GAS.MASS;
COST = 0.0;

```

Save the exit flow.

```

FLC = GAS;
RETURN;
GT1: PROC(P_OUT,EFF);
  DCL (P_OUT,EFF) FLOAT(16);

```

Set the gas pressure equal to the exit flow pressure.

```
GAS.PRES=P_OUT;
```

Call the properties code, with the pressure and entropy as the specified inputs, to determine the isentropic-expansion conditions of the flow.

```
CALL GP(NAME,GAS,11B);
```

Using the isentropic state point and the specified efficiency of the expansion, set the exit enthalpy of the flow.

```
GAS.ENTH = FLC.ENTH-(FLC.ENTH-GAS.ENTH)* EFFICIENCY;
```

Call the properties code, with the pressure and enthalpy specified, to determine the exit conditions of the flow.

```
CALL GP(NAME,GAS,10B);
END GT1;

GTIN: ENTRY(GT_P);
IF MODE='DESIGN' THEN
  CALL TABIN(PRATIO,DDNAME);
RETURN;

GTOUT: ENTRY(GT_P);
PUT SKIP EDIT(T',NAME)(COL(4),A);
PUT SKIP(2) EDIT('MODE = ',MODE)(COL(10),A,A)
('EXIT PRESSURE = ',EXIT_PRES,
'EFFICIENCY = ',EFFICIENCY,
'MECHANICAL EFFICIENCY = ',MECH_EFF,
'MASS FACTOR = ',MASS_FACT,
'M FACTOR = ',M_FACT,
'DESIGN PRESSURE RATIO = ',PRDES,
'PRESSURE RATIO = ',PRES_RATIO)
(COL(10),A,E(12,5));
IF M_FACT>1.0 THEN
  PUT SKIP(2) EDIT('FLOW IS CHOKED, M_FACT=',M_FACT)
(COL(10),A,E(12,5));
END GTC;
```

A.9 HEATER MODEL

A.9.1 Description of Model

The heater model (HT) requires one pass-through flow of the generic type GAS. The parameters of the model are:

HEAT -- Specified heat added to the flow.

T_SET -- Specified exit temperature of the flow if set to a number greater than zero. This exit temperature determines the value of HEAT; if T_SET is set to zero, then HEAT must be put in.

PRES_DROP_FRAC -- Fraction of the input flow pressure used as a pressure drop through the heater.

A.9.2 Declaration Structure

```

* PROCESS NAME('HTC');
HTC: PROC( HT_P, FLOW_P);

  DCL (HT_P, FLOW_P) POINTER;
  DCL 1 HT BASED(HT_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP, PRES, ENTH, ENTP, QUAL, RHO, VEL, MASS) FLOAT(16),
    3 COMP,
    4 (XAR, XCH4, XCO, XCO2, XH, XH2, XH2O, XH2S, XK, XKOH, XNO, XN2,
      XO, XOH, XO2, XSO2, XHCL, XCH3OH, XC, XCOS, XNH3, XS, XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
  2 PARM,
    3 HEAT FLOAT(16),
    3 T_SET FLOAT(16),
    3 PRES_DROP_FRAC FLOAT(16),
  2 POWER,
    3 (INPUT, PRODUCED, CONSUMED, LOSS) FLOAT(16);
  DCL FLOW BASED(FLOW_P) LIKE FLC;
  DCL (GP) ENTRY,
    (X) FLOAT(16);

```

Initialize the power to zero and set the exit flow pressure using any prescribed pressure-drop fraction.

```

POWER=0.0;
FLOW.PRES=FLOW.PRES*(1.0-PRES_DROP_FRAC);

```

Save the inlet enthalpy of the flow for calculating the heat transferred if the T_SET parameter is set.

```

X=FLOW.ENTH;
IF T_SET=0.0 THEN
  DO;

```

If T_SET is not zero, use this value of the exit temperature to calculate the exit enthalpy, and thus the heat transferred.

```

  FLOW.TEMP=T_SET;
  CALL GP(NAME, FLOW, 1B);
  HEAT=(FLOW.ENTH-X)*FLOW.MASS;
END;

```

Using the heat transferred, set the exit flow enthalpy.

```

FLOW.ENTH=X+HEAT/FLOW.MASS;

```

Call the properties code to determine the other state variables of the flow at the exit.

```
CALL GP(NAME, FLOW, 10B);
```

Since the heat transferred within the HT model does not come from any flow within the model, the heat transferred represents additional thermal input (or loss) to the system. This heat is added to the POWER substructure.

```
IF HEAT >= 0.0 THEN
  POWER.INPUT = HEAT;
ELSE
  POWER.LOSS = -HEAT;
```

Save the exit flow.

```
FLC = FLOW;
RETURN;
```

```
HTOUT: ENTRY( HT_P );
  PUT SKIP EDIT(' ', NAME)(COL(4), A);
  PUT SKIP(2) EDIT('HEAT = ', HEAT)(COL(10), A, E(12, 5));
END HTC;
```

A.10 HEAT-EXCHANGER MODEL

A.10.1 Description of Model

The heat-exchanger model (HX) is set up to process the hot flow in one entry (HXH) and the cold flow in another entry (HXC). Both entries require one pass-through flow of the generic type GAS. Either entry may be called first.

The model also includes options for calculating heat-transfer-surface areas using specified heat-transfer coefficients. These coefficients can be adjusted as functions of mass flow rate or temperature to simulate off-design changes. Thus, the model can be run off-design, but the surface areas (as calculated in the code) must be constrained to equal their design values outside the model (see CONS, below).

The parameters of the HX model are as follows:

MODE -- Specified character string taking the values of "DESIGN" or "OFF-DESIGN."

TYPE -- Specified character string taking the values "PARALLEL" or "COUNTER" to indicate that the heat exchanger is of either a parallel-flow or counterflow configuration.

HEAT -- Specified heat transfer from the hot to the cold fluid (may be overridden if T_SET is set).

HEAT_FLUX -- Calculated average heat flux in the exchanger.

T_SET(2) -- An array of exit-temperature values: the first element is for the hot flow; the second, for the cold flow. Only one of these elements should be assigned a value (their default value is zero), and that one must correspond to the entry that is called first. If either element is assigned a value, then the heat transferred is calculated from the T_SET value rather than from the value set in HEAT; T_SET should be used only if the flow being assigned an exit temperature is definitely not in the two-phase region.

PRES_DROP_FRAC(2) -- Specified array of the fraction of input flow pressure used as a pressure drop through the device: hot flow, 1, and cold flow, 2.

U -- Calculated overall heat-transfer coefficient from the hot to the cold fluids.

AREA -- Total heat-transfer area, calculated in the design mode and specified in the off-design mode (however, see CONS).

INTEMP(2) -- Storage for the inlet fluid temperatures.

AVGTEMP(2) -- Calculated average temperatures of the hot and cold fluids.

ST(2) -- Calculated surface temperatures between the fluids and the wall.

LMTD -- Calculated log mean temperature difference between the fluids.

UR(3) -- Specified array of heat-transfer coefficients for hot fluid to wall (1), wall to cold fluid (2), and through wall (3).

UC(2) -- Specified array of correction factors for the UR(1) and UR(2) values in the off-design mode. If a value of UC exceeds 100, then the value of UR is adjusted as $UR \cdot (UC / AVGTEMP)^3$ or else as $UR \cdot (DM / MASS)^{UC}$, where MASS is the fluid-mass flow rate and DM is defined below. These parameters are used only in the off-design mode.

DM(2) -- Specified input values of the design mass flow rates (used only in the off-design mode, to correct the heat-transfer coefficients).

CONS -- Calculated off-design parameter representing the difference between the calculated and specified surface areas. In the off-design mode, this variable must be constrained to equal zero if the model is to yield the correct results.

PINCH_POINT -- Specified parameter representing the minimum value of the MEAN_TDIF for which no error message indicating occurrence of a pinch-point violation is printed.

CAL(2) -- "Flags" used by the code to indicate when both hot and cold entries have been called; these flags indicate when the surface areas are to be calculated.

A.10.2 Declaration Structure

```
* PROCESS NAME('HXH');
HXH: PROC( HX_P, FLOW_P);

DCL (HX_P, FLOW_P) POINTER;
DCL 1 HX BASED(HX_P),
    2 NAME CHAR(16),
    2 FLH,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP, PRES, ENTH, ENTP, QUAL, RHO, VEL, MASS) FLOAT(16),
    3 COMP,
    4 (XAR, XCH4, XCO, XCO2, XH, XH2, XH2O, XH2S, XK, XKO, XNO, XN2,
      XO, XOH, XO2, XSO2, XHCL, XCH3OH, XC, XCOS, XNH3, XS, XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP, PRES, ENTH, ENTP, QUAL, RHO, VEL, MASS) FLOAT(16),
    3 COMP,
    4 (XAR, XCH4, XCO, XCO2, XH, XH2, XH2O, XH2S, XK, XKO, XNO, XN2,
      XO, XOH, XO2, XSO2, XHCL, XCH3OH, XC, XCOS, XNH3, XS, XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 MODE CHAR(10),
    3 TYPE CHAR(18),
    3 HEAT FLOAT(16),
    3 HEAT FLUX FLOAT(16),
    3 T SET(2) FLOAT(16),
    3 PRES DROP FRAC(2) FLOAT(16),
    3 U FLOAT(16),
    3 AREA FLOAT(16),
    3 INTEMP(2) FLOAT(16),
    3 AVGTEMP(2) FLOAT(16),
    3 ST(2) FLOAT(16),
    3 LMTD FLOAT(16),
```

```

      3 UR(3) FLOAT(16),
      3 UC(2) FLOAT(16),
      3 DM(2) FLOAT(16),
      3 CONS FLOAT(16),
      3 PINCH_POINT FLOAT(16),
      3 FIRST_FLOW FIXED BIN(15),
      3 CAL(2) FIXED BIN(15);
DCL FLOW BASED(FLOW_P) LIKE FLH;
DCL (GP) ENTRY,
      (URT(3),X,Y,Z,H1,H2,HMIN) FLOAT(16),
      (I, FLOW_#) FIXED BIN(15);

```

If the hot entry has been called, set the flow number to 1 and call HXWK to perform the calculations.

```

FLOW_#=1;
CALL HXWK(FLOW_#);
RETURN;

```

```

HXC: ENTRY( HX_P, FLOW_P);

```

If the cold entry has been called, set the flow number to 2 and call HXWK to perform the calculations.

```

FLOW_#=2;
CALL HXWK(FLOW_#);
RETURN;
HXWK: PROC(FLOW_#);
DCL FLOW_# FIXED BIN(15);

```

Save the inlet value of temperature for later use in calculating the log mean temperature.

```

INTEMP(FLOW_#)=FLOW.TEMP;

```

If the mode is design, then save the inlet mass flow rate for printout with the model outputs.

```

IF MODE='DESIGN' THEN
  DM(FLOW_#)=FLOW.MASS;

```

Set the exit pressure of the flow using the specified pressure-drop fraction, adjusted by the ratio of the actual-to-design mass flow rates squared. (If the mode is design, this ratio is equal to 1 and does not affect the pressure drop.)

```

FLOW.PRES=FLOW.PRES*(1.0-PRES_DROP_FRAC(FLOW_#)*
  (FLOW.MASS/DM(FLOW_#))**2);

```

Temporarily save the inlet flow enthalpy.

```

X=FLOW.ENTH;

```

If the T_SET value for this flow is not equal to zero, then set the flow's exit temperature to this value and calculate the heat transferred in order to reach this temperature. (This option can be used only when the flow at the exit is known to be in a single phase.)

```

IF T_SET(FLOW_#)=0.0 THEN
DO;
  FLOW.TEMP=T_SET(FLOW_#);
  CALL GP(NAME, FLOW, 1B);
  IF FLOW_#=1 THEN
    HEAT=(X-FLOW.ENTH)*FLOW.MASS;
  ELSE
    HEAT=(FLOW.ENTH-X)*FLOW.MASS;
END;

```

Set the value of CAL for this flow to 1 to indicate that the model has been called for this entry.

```

CAL(FLOW_#)=1;

```

Using the value of HEAT, calculate the exit enthalpy of the flow and call the properties code to obtain the other exit flow conditions.

```

IF FLOW_#=1 THEN
  FLOW.ENTH=X-HEAT/FLOW.MASS;
ELSE
  FLOW.ENTH=X+HEAT/FLOW.MASS;
CALL GP(NAME, FLOW, 10B);

```

Calculate the average temperature of the flow.

```

AVGTEMP(FLOW_#)=0.5*(FLOW.TEMP+INTEMP(FLOW_#));

```

Save the exit flow.

```

IF FLOW_#=1 THEN
  FLH=FLOW;
ELSE
  FLC=FLOW;

```

If both CAL(1) and CAL(2) are equal to 1, then both sides of the heat exchanger have been called, and the calculation of the log mean temperature and heat-transfer-surface areas can be made.

```

IF CAL(1)=1 & CAL(2)=1 THEN
DO;

```

Depending on the type of heat exchanger, set the difference between the entering and leaving flow temperatures in x and y.

```

SELECT;
  WHEN(INDEX(TYPE, 'PARAL')=0)

```

```

DO;
  X=INTEMP(1)-INTEMP(2);
  Y=FLH.TEMP-FLC.TEMP;
END;
WHEN(INDEX(TYPE, 'COUNT')=0)
DO;
  X=INTEMP(1)-FLC.TEMP;
  Y=FLH.TEMP-INTEMP(2);
END;
END;

```

Using the differences in terminal temperatures, calculate the log mean temperature difference. At this point in the calculations, it may be that the temperature profiles cross; if so, a fictitious negative log mean temperature difference is calculated for these cases. Note the log mean temperature difference is a continuous function of x and y.

```

SELECT;
  WHEN(X>0.0 & Y>0.0 & LOG(X/Y)=0.0)
    LMTD=(X-Y)/LOG(X/Y);
  WHEN(X>0.0 & Y>0.0 & LOG(X/Y)=0.0)
    LMTD=X;
  WHEN(Y<=0. & X>=0.)
    LMTD=-SQRT(ABS(Y));
  WHEN(X<=0. & Y>=0.)
    LMTD=-SQRT(ABS(X));
  WHEN(X<0. & Y<0.)
    LMTD=-(X**2+Y**2)**0.25;
END;

```

Call CALU to obtain the heat-transfer coefficient.

```
CALL CALU;
```

If the mode is design, calculate the surface area; otherwise, calculate the difference between the heat flux for the required surface area and that calculated by the model.

```

IF MODE='DESIGN' THEN
  IF LMTD=0. THEN
    AREA=HEAT/(U*LMTD);
  ELSE
    AREA=1E60;
ELSE
  CONS=HEAT/AREA-U*LMTD;

```

Calculate the value of the heat flux and wall temperatures for printout.

```

HEAT FLUX=HEAT/AREA;
ST(1)=INTEMP(1)-UR(1)*U*LMTD;
ST(2)=ST(1)-UR(3)*U*LMTD;
IF FLOW #1 THEN
  FLOW=FLH;
ELSE
  FLOW=FLC;

```

Reset the values of CAL to 0.

```
CAL=0;
END;
END HXWK;
CALU: PROC;
```

Calculate the value of the heat-transfer coefficient, either by setting the individual film coefficients to those specified (if in the design mode) or by adjusting the specified film coefficients using UC.

```
IF MODE='DESIGN' THEN
  URT=UR;
ELSE
  DO;
    URT(3)=UR(3);
    IF UC(1)>100. THEN
      URT(1)=UR(1)*(UC(1)/AVGTEMP(1))*3;
    ELSE
      URT(1)=UR(1)*(DM(1)/FLH.MASS)**UC(1);
    IF UC(2)>100. THEN
      URT(2)=UR(2)*(UC(2)/AVGTEMP(2))*3;
    ELSE
      URT(2)=UR(2)*(DM(2)/FLC.MASS)**UC(2);
    END;
    U=1.0/(URT(1)+URT(2)+URT(3));
  END CALU;
HXOUT: ENTRY( HX P );
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('MODE = ',MODE)(COL(10),A,A)
  ('TYPE = ',TYPE)(COL(10),A,A)
  ('DESIGN MASS FLOW RATES = ',DM,' KG/S')(COL(10),A,2 F(9,2),A)
  ('INLET TEMPERATURES = ',INTEMP,' K')(COL(10),A,2 F(8,2),A)
  ('AVERAGE TEMPERATURES = ',AVGTEMP,' K')(COL(10),A,2 F(8,2),A)
  ('DESIGN THERMAL RESISTIVITIES = ',UR,' SQ-M K/W')
  (COL(10),A,3 E(13,4),A);
IF MODE='DESIGN' THEN
  DO;
    CALL CALU;
    PUT SKIP EDIT('THERMAL RESISTIVITIES = ',URT,' SQ-M K/W')
      (COL(10),A,3 E(13,4),A)
      ('RESISTANCE CORRECTIONS = ',UC)
      (COL(10),A,2 E(13,4));
  END;
PUT SKIP EDIT('OVERALL HEAT TRANSFER COEF = ',U,' W/SQ-M K')
  (COL(10),A,E(12,5),A)
  ('LOG MEAN TEMP DIFFERENCE = ',LMTD,' K')
  (COL(10),A,E(12,5),A)
  ('HEAT TRANSFERRED = ',HEAT,' W')
  (COL(10),A,E(12,5),A)
  ('HEAT TRANSFER SURFACE AREA = ',AREA,' SQ-M')
  (COL(10),A,E(12,5),A)
  ('HEAT FLUX = ',HEAT_FLUX,' W/SQ-M')(COL(10),A,E(12,5),A)
```

```

      ('SURFACE TEMPERATURES = ',ST,' K')(COL(10),A,2 F(8,2),A);
      IF LMTD<PINCH POINT THEN
      PUT SKIP EDIT(' *** PINCH POINT CONSTRAINT VIOLATED')
      (COL(4),A);
      END HXH;

```

A.11 FLOW-INITIATOR MODEL

A.11.1 Description of Model

The flow-initiator model (IN) requires one pass-through flow of the generic type GAS. The IN model also has two additional GAS flow-processing entries, INCYCL and INCOMP. The INCYCL entry calculates the differences in temperature, pressure, etc. of the flow between this entry and that of the INC entry. This entry is useful in setting up recycle loops.

The INCOMP entry is used to feed back to the INC entry the values of the INCOMP entry's GAS flow compositions. By calling this entry with the parameter ITER (incremented by one for each call), INCOMP will take its input flow composition and assign it to the model's COMP parameter. In this way, a simple fixed-point iteration scheme can be set up to converge on gas compositions in recycle loops by sweeping ITER from one to some maximum iteration number, calling INC at the beginning of the loop and INCOMP at the end of the loop.

The parameters of the IN model are as follows:

ID -- Specified character-string variable representing the type of properties code used in calculating thermodynamic properties of the flow.

ATOM(8) -- Calculated array of atomic-weight fractions of elements of the flow, if the ID is set to GAS.

T -- Specified temperature of the flow. If T is set to zero, the flow is assumed to be a condensible fluid and the saturation temperature is used. In this case, the enthalpy of the flow is determined using Q.

P -- Specified pressure of the flow.

H -- Calculated enthalpy of the flow.

S -- Calculated entropy of the flow.

Q -- Specified quality of the flow, used when T is set to zero. Flow quality Q may be greater than one (to represent superheating) or less than zero (to represent subcooling).

V -- Specified velocity of the flow.

M -- Specified mass flow rate of the flow.

COMP -- Specified structure variable defining the molar fractions of the separate species that may be used with the GAS properties code. The molar fraction of each species is specified as "X," followed by the species' chemical formula (e.g., XH₂, XCO₂, XNH₃).

SOL -- Structure variable representing the weight fractions of entrained solids within the gas flow. At present, SOL has only a single scalar (WTF) within its structure. This WTF represents the fraction of the flow's mass flow rate that is solid.

DT, DP, DV, DH, and DM -- Calculated differences in T, P, V, H, and M between the flow originating from the INC entry and the flow entering the INCYCL entry.

ACC -- Termination criterion employed when the INCOMP entry and a SALT-defined parameter sweep over ITER are used to close a recycle loop over compositions. If the maximum difference in species concentrations between the INC and INCOMP entries is less than ACC, then ITER is set to 1000.

ITER -- Iteration counter used in the INCOMP entry.

ITERS -- Saved previous value of ITER.

PRINT -- Print switch used in the INCOMP entry.

A.11.2 Declaration Structure

```
* PROCESS NAME('INC');
INC: PROC(IN_P, FLOW_P);

DCL (IN_P, FLOW_P) POINTER;
DCL 1 IN BASED(IN_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP, PRES, ENTH, ENTP, QUAL, RHO, VEL, MASS) FLOAT(16),
    3 COMP,
    4 (XAR, XCH4, XCO, XCO2, XH, XH2, XH2O, XH2S, XK, XKOH, XNO, XN2,
    XO, XOH, XO2, XSO2, XHCL, XCH3OH, XC, XCOS, XNH3, XS, XCL) FLOAT(16),
    3 SOL,
```

```

      4 WTF FLOAT(16),
2  PARM,
      3 FNAME CHAR(16),
      3 ID CHAR(16) VARYING,
      3 ATOM(8) FLOAT(16),
      3 (T,P,H,S,Q,R,V,M) FLOAT(16),
      3 COMP,
          4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
            XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
      3 SOL,
          4 WTF FLOAT(16),
      3 (DT,DP,DV,DH,DM) FLOAT(16),
      3 ACC FLOAT(16),
      3 (ITER,ITERS,PRINT) FIXED BIN(15),
2  POWER,
      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL FLOW BASED(FLOW_P) LIKE FLC;
DCL (GP,GPSAT,GASBW) ENTRY;
DCL (PC,HL,HS) FLOAT(16);
DCL I FIXED BIN(15);

```

Set POWER to zero. (Note: POWER is only redefined if the CYCL entry is called, in which case POWER.INPUT will represent the enthalpy needed to close a loop.)

```
POWER=0.0;
```

Using the input parameter values, initialize the flow.

```

FLOW.PRES=PARM.P;
FLOW.VEL=PARM.V;
FLOW.MASS=PARM.M;
FLOW.ID=PARM.ID;
FLOW.COMP=PARM.COMP;

```

If the flow ID is that of "GAS," then call GASBW to obtain the kg-atoms/kg array. These values are saved in PARM.ATOM, as well as in FLOW.ATOM.

```

IF FLOW.ID='GAS' THEN
  CALL GASBW(FLOW.COMP,PARM.ATOM);
FLOW.ATOM=PARM.ATOM;
FLOW.SOL=PARM.SOL;
IF PARM.T=0.0 THEN
  DO;

```

If the specified temperature is zero, it is assumed that the flow is one of the condensible fluids and GPSAT is called to obtain the saturation values of the enthalpy. These values are then used along with the specified flow quality to obtain the enthalpy of the flow.

```

CALL GPSAT(NAME,FLOW,PC,HL,HS);
FLOW.ENTH=HL+PARM.Q*(HS-HL);

```

Call the properties code to obtain the exit flow conditions.

```
      CALL GP(NAME, FLOW, 10B);
      END;
    ELSE
      DO;
```

If the flow temperature has been set, call the properties code to obtain the flow's enthalpy.

```
      FLOW.TEMP=PARM.T;
      CALL GP(NAME, FLOW, 1B);
      PARM.H=FLOW.ENTH;
    END;
```

Save the values of the species' molar fractions. These are really used only when the flow ID is that of "GAS." The inlet values of the species' molar fractions are redefined at this point; thus, PARM.COMP should probably not be used as parameters to be varied to establish system constraints.

```
      PARM.COMP=FLOW.COMP;
      FLC=FLOW;
      RETURN;
```

```
    INCYCL: ENTRY(IN_P, FLOW_P);
```

Calculate the value of any input power necessary to close the loop. If the loop is closed by means of a system constraint, this power should go to zero.

```
      POWER.INPUT=FLC.MASS*(FLC.ENTH+0.5*FLC.VEL**2)-
        FLOW.MASS*(FLOW.ENTH+0.5*FLOW.VEL**2);
```

Calculate the differences between the flow variables initiated by the model and those coming into this CYCL entry.

```
      DT=FLOW.TEMP-FLC.TEMP;
      DP=FLOW.PRES-FLC.PRES;
      DV=FLOW.VEL-FLC.VEL;
      DH=FLOW.ENTH-FLC.ENTH;
      DM=FLOW.MASS-FLC.MASS;
      RETURN;
```

```
    INCOMP: ENTRY(IN_P, FLOW_P);
```

INCOMP is an entry that can be used to perform a simple fixed-point iteration on the species' molar fractions when a recycle loop is required on gas flows. This is accomplished by setting up a SYSBEG/SYSEND SWEEP loop with the IN model at the beginning and the INCOMP entry at the end and sweeping over IN.ITER. When the ATOM arrays of the initiated and incoming flows are less than IN.ACC, ITER is set to 1000 to terminate the sweep.

```

IF ITER>ITERS THEN
DO;
  IF ITER<=1 THEN
    ITERS=0;
  IF PRINT>=1 THEN
    DO;
      PUT SKIP EDIT('RECYCL: ',NAME,' N=',PARM.ITER)
        (SKIP,COL(2),A,A,A,F(3),COL(10),A);
      PUT EDIT('FLC.ATOM= ',FLC.ATOM,'FLOW.ATOM=',FLOW.ATOM)
        (SKIP,COL(5),A,8 E(12,4));
    END;
  IF ALL(ABS(FLOW.ATOM-FLC.ATOM)<ACC) THEN
    DO;
      ITER=1000;
      RETURN;
    END;
  PARM.COMP=FLOW.COMP;
END;
RETURN;

INOUT: ENTRY(IN P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('ID=',FLC.ID)(COL(10),A,A)
  ('TEMP = ',FLC.TEMP,'PRES = ',FLC.PRES,'VEL = ',FLC.VEL,
  'ENTH = ',FLC.ENTH,'MASS = ',FLC.MASS)
  (COL(10),A,E(12,5));
END INC;

```

A.12 FUEL-FLOW-INITIATOR MODEL

A.12.1 Description of Model

The fuel-flow-initiator model (INF) requires one pass-through flow of the generic type FUEL. The parameters of the INF model are as follows:

T -- Specified temperature of the fuel.

M -- Specified mass flow rate of the fuel.

HHV -- Specified higher heating value of the fuel.

WEIGHTS -- Structure variable representing the fuel composition by weight fractions. The WEIGHTS structure includes the following variables, where each variable represents the weight fraction of the substance in parentheses: C (carbon), H (hydrogen), O (oxygen), N (nitrogen), S (sulfur), Cl (chlorine), H₂O (water), and ASH (ash).

A.12.2 Declaration Structure

```
* PROCESS NAME('INFC');
  INFC: PROC(INF_P, FUEL_P);

    DCL (INF_P, FUEL_P) POINTER;
    DCL 1 INF BASED(INF_P),
      2 NAME CHAR(16),
      2 PARM,
      3 PROP,
      4 (TEMP,MASS,HHV) FLOAT(16),
      3 WEIGHTS,
      4 (C,H,O,N,S,CL,H2O,ASH) FLOAT(16),
      2 POWER,
      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
    DCL 1 FUEL BASED(FUEL_P),
      2 FNAME CHAR(16),
      2 PROP,
      3 (TEMP,MASS,HHV) FLOAT(16),
      2 WEIGHTS,
      3 (C,H,O,N,S,CL,H2O,ASH) FLOAT(16);
```

Initialize power based on the higher heating value of the fuel.

```
POWER=0.0;
POWER.INPUT=PARM.MASS*PARM.HHV;
```

Initialize fuel flow using the specified model inputs.

```
FUEL.PROP=PARM.PROP;
FUEL.WEIGHTS=PARM.WEIGHTS;
RETURN;

INFOUT: ENTRY(INF_P);
  PUT SKIP(2) EDIT(' ',NAME)(COL(4),A)
    ('FUEL HHV=',PARM.HHV,'FUEL MASS=',PARM.MASS)(COL(10),A,E(12,5));
  PUT SKIP(2) EDIT('FUEL WEIGHT FRACTIONS')(COL(10),A)
    ('CARBON','HYDROGEN','OXYGEN','NITROGEN','SULFUR','CHLORINE',
    'WATER','ASH',PARM.WEIGHTS)(COL(15),8 A(10),COL(13),8 F(10,6));
  END INFC;
```

A.13 MOLTEN-CARBONATE FUEL-CELL MODEL

A.13.1 Description of Model

The molten-carbonate fuel-cell model (MCFC) requires two pass-through flows of the generic type GAS. The first of these is the anode flow, and the second is the cathode flow.

The parameters of the MCFC model are as follows:

CELL_CURRENT -- Specified current through each cell.

CELL_VOLTAGE -- Calculated cell voltage.

CELL_TEMP -- Specified average temperature of a cell.

STACK_VOLTAGE - Calculated total voltage across the cell stack.

NO_OF_CELLS -- Specified total number of cells in the stack.

DELTA_VOLT -- Specified difference between the Nernst potential at the fuel-cell exit and the cell voltage.

FUEL_UTIL -- Calculated value of the fuel utilization.

O2_UTIL -- Calculated value of the O₂ utilization.

CO2_UTIL -- Calculated value of the CO₂ utilization.

HF -- Calculated value of the overall isothermal heat of reaction.

E -- Calculated Nernst potential at the fuel-cell exit.

A.13.2 Declaration Structure

```
* PROCESS NAME('MCFCC');
```

```
  MCFCC: PROC(MCFC_P,FGAS1_P,FGAS2_P);
```

```
  DCL (MCFC_P,FGAS1_P,FGAS2_P) POINTER;
```

```
  DCL 1 MCFC BASED(MCFC_P),
```

```
    2 NAME CHAR(16),
```

```
    2 FLC1,
```

```
      3 FNAME CHAR(16),
```

```
      3 ID CHAR(16) VARYING,
```

```
      3 ATOM(8) FLOAT(16),
```

```
      3 PROP,
```

```
        4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
```

```
      3 COMP,
```

```
        4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
```

```
        XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
```

```
      3 SOL,
```

```
        4 WTF FLOAT(16),
```

```
    2 FLC2,
```

```
      3 FNAME CHAR(16),
```

```
      3 ID CHAR(16) VARYING,
```

```
      3 ATOM(8) FLOAT(16),
```

```
      3 PROP,
```

```

      4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
      4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
      4 WTF FLOAT(16),
2 PARM,
      3 CELL_CURRENT          FLOAT(16),
      3 CELL_VOLTAGE          FLOAT(16),
      3 STACK_VOLTAGE         FLOAT(16),
      3 CELL_TEMP             FLOAT(16),
      3 NO_OF_CELLS           FLOAT(16),
      3 DELTA_VOLT            FLOAT(16),
      3 FUEL_UTIL              FLOAT(16),
      3 O2_UTIL                FLOAT(16),
      3 CO2_UTIL               FLOAT(16),
      3 (HF,E)                 FLOAT(16),
2 POWER,
      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL FGAS1 BASED(FGAS1_P) LIKE FLC1,
   FGAS2 BASED(FGAS2_P) LIKE FLC2;
DCL (Y,CO3_MOL,CO2_MOL,O2_MOL,H2_MOL,F,H_DEL_TC,MOL_WT,
     H_DEL,TTRY,HAIN,HAIN_TC,HAOUT,HAOUT_TC,
     EO,P_H2O_AN,P_CO2_AN,P_H2_AN,P_CO2_CA,P_O2_CA,
     HGIN,HGIN_TC,HGOUT,HGOUT_TC,GROSS_POWER,CH4_REQ) FLOAT(16);
DCL (GASBW,GASMR,GP) ENTRY;
DCL SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
              FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*));

```

Perform the anode calculations first: Set Faraday's constant and initialize the power to zero.

```

F=96487.0;
POWER=0.0;

```

Save the value of the inlet anode flow enthalpy.

```

HAIN = FGAS1.ENTH*FGAS1.MASS;

```

Set the anode flow temperature to the specified cell temperature and call the properties code to obtain the state conditions of the flow at this temperature.

```

FGAS1.TEMP = CELL_TEMP;
CALL GP(NAME,FGAS1,1B);

```

Save the value of the anode flow enthalpy at this cell temperature.

```

HAIN_TC = FGAS1.ENTH*FGAS1.MASS;

```

Since the fuel cell only works with the gas flow streams, any entrained-solid flow is temporarily subtracted out and saved in the FLC1 flow variables.

```

FLC1.WTF=FGAS1.WTF*FGAS1.MASS;
FLC1.MASS=FGAS1.MASS-FLC1.WTF;

```

The procedure GASMR will calculate the molar flow rates of the flow, given the mass flow rate and the species' molar fractions, if its first argument is 1. These molar flow rates are temporarily stored in the FLC1 variables.

```

CALL GASMR(1B,FGAS1.COMP,FLC1.MASS,FLC1.COMP,MOL_WT);

```

Given the cell current and number of cells, calculate the total number of moles of CO₂ and O₂ that will cross over from the cathode stream. These species are then added to those already within the anode stream.

```

CO3_MOL = CELL_CURRENT*NO_OF_CELLS/(2000.0*F);
CO2_MOL = CO3_MOL;
O2_MOL = CO3_MOL/2.0;
FLC1.XCO2 = FLC1.XCO2+CO2_MOL;
FLC1.XO2 = FLC1.XO2+O2_MOL;

```

Calculate the hydrogen concentration within the anode flow.

```

H2_MOL = 4.0*FLC1.XCH4+FLC1.XH2+FLC1.XCO;

```

Check whether or not there is sufficient hydrogen concentration within the anode flow. If not, indicate how much more is needed and terminate.

```

IF (H2_MOL < CO3_MOL) THEN
DO;
  CH4_REQ = CO3_MOL*1.25/4.0 - FLC1.XH2;
  PUT SKIP(2) EDIT('** ERROR IN MCFC:',
    'NOT ENOUGH FUEL IN ANODE INLET STREAM.',
    'INCREASE CH4 FLOW RATE TO ',CH4_REQ,
    '(MOL/S) OR MORE.')
    (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,
    E(13,5),X(2),A);
  STOP;
END;

```

Calculate the fuel utilization.

```

FUEL_UTIL = CO3_MOL/H2_MOL;

```

Recall the GASMR procedure, with its first argument set to 2, in order to convert the molar flow rates back to species molar fractions and to obtain the total mass flow rate. These values are placed into the FGAS1 flow variables.

```

CALL GASMR(10B,FGAS1.COMP,FGAS1.MASS,FLC1.COMP,MOL_WT);

```

Given the new values of the FGAS1 composition, call GASBW to obtain the ATOM array.

```

CALL GASBW(FGAS1.COMP,FGAS1.ATOM);

```

Add any entrained solids back into the flow and call the properties code to obtain the exit flow conditions.

```
FGAS1.MASS=FGAS1.MASS+FLC1.WTF;
FGAS1.WTF=FLC1.WTF/FGAS1.MASS;
CALL GP(NAME,FGAS1,1B);
```

Save the exit flow enthalpy from the anode.

```
HAOUT_TC = FGAS1.ENTH*FGAS1.MASS;
```

Now perform the cathode calculations: Save the cathode inlet enthalpy.

```
HCIN = FGAS2.ENTH*FGAS2.MASS;
```

Set the cathode flow temperature to that of the cell temperature.

```
FGAS2.TEMP = CELL_TEMP;
```

Call the properties code to obtain the state conditions of the cathode flow at the cell temperature and save the value of its enthalpy.

```
CALL GP(NAME,FGAS2,1B);
HCIN_TC = FGAS2.ENTH*FGAS2.MASS;
```

Subtract out any entrained solids and call GASMR to obtain the molar flow rates (as was done for the anode stream).

```
FLC2.WTF=FGAS2.WTF*FGAS2.MASS;
FLC2.MASS=FGAS2.MASS-FLC2.WTF;
CALL GASMR(1B,FGAS2.COMP,FLC2.MASS,FLC2.COMP,MOL_WT);
```

Adjust the species molar rates to reflect the crossover of CO_2 and O_2 and calculate the CO_2 and O_2 utilization rates.

```
CO2_UTIL = CO2_MOL/FLC2.XCO2;
O2_UTIL = O2_MOL/FLC2.XO2;
FLC2.XCO2 = FLC2.XCO2-CO2_UTIL;
FLC2.XO2 = FLC2.XO2-O2_UTIL;
```

Check whether or not there is sufficient CO_2 in the cathode stream. If not, indicate how much would be needed and terminate.

```
IF (FLC2.XCO2 < 0.0) THEN
DO;
  PUT SKIP(2) EDIT('** ERROR IN MCFC:',
    'NOT ENOUGH CO2 IN CATHODE INLET STREAM.',
    'INCREASE CO2 MOLE RATE TO ',1.25*CO2_MOL,
    '(MOL/S) OR MORE')
    (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,E(13,5),
    X(2),A);
```

```

      STOP;
    END;

```

Check whether or not there is sufficient O_2 in the stream. If not, indicate how much would be needed and terminate.

```

    IF (FLC2.XO2 < 0.0) THEN
      DO;
        PUT SKIP(2) EDIT('** ERROR IN MCFC:',
          'NOT ENOUGH O2 IN CATHODE INLET STREAM.',
          'INCREASE O2 MOLE RATE TO ',1.25*O2_MOL,
          '(MOL/S) OR MORE')
          (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,E(13,5),
            X(2),A);
        STOP;
      END;

```

Reconvert back to molar fractions, set the new ATOM array values, add the subtracted entrained solids, and call the properties code.

```

    CALL GASMR(10B,FGAS2.COMP,FGAS2.MASS,FLC2.COMP,MOL_WT);
    CALL GASBW(FGAS2.COMP,FGAS2.ATOM);
    FGAS2.MASS=FGAS2.MASS+FLC2.WTF;
    FGAS2.WTF=FLC2.WTF/FGAS2.MASS;
    CALL GP(NAME,FGAS2,1B);

```

Save the exit cathode flow enthalpy.

```

    HCOUT_TC = FGAS2.ENTH*FGAS2.MASS;

```

Finally, perform the energy-balance calculations: Save the total enthalpy change across the cell (at the cell temperature).

```

    H_DEL_TC=HAOUT_TC-HAIN_TC+HCOUT_TC-HCIN_TC;
    HF=-H_DEL_TC;

```

The actual exit temperature of the cell will be determined by varying the exit temperature until the total enthalpy change across the cell stack represents the gross power produced by the cell. The first try at the exit temperature is taken as that of the cell temperature. Thereafter, exit temperature is controlled by the one-dimensional equation solver, SOV.

```

    TTRY=CELL_TEMP;
    DO I=1 TO 25;

```

Set the anode exit temperature and call the properties code to obtain the exit enthalpy.

```

    FGAS1.TEMP = TTRY;
    CALL GP(NAME,FGAS1,1B);
    HAOUT = FGAS1.ENTH*FGAS1.MASS;

```

Set the cathode exit temperature and call the properties code to obtain the exit enthalpy.

```
FGAS2.TEMP = TTRY;
CALL GP(NAME,FGAS2,1B);
HCOUT = FGAS2.ENTH*FGAS2.MASS;
```

Calculate the total enthalpy change across the cell.

```
H_DEL=HAOUT-HAIN+HCOUT-HCIN;
```

The gross power output from the cell depends on the stack voltage, and hence, on the cell voltage. The cell voltage is taken as the Nernst potential minus some specified voltage drop, DELTA_VOLT. The Nernst potential is a function of the partial pressures of the species at the exit of the anode and cathode streams. These are calculated first, followed by the Nernst potential, cell voltage, and stack voltage, and then by the gross power output.

```
P_H2O_AN = FGAS1.XH2O*FGAS1.PRES;
P_CO2_AN = FGAS1.XCO2*FGAS1.PRES;
P_H2_AN = FGAS1.XH2*FGAS1.PRES;
P_CO2_CA = FGAS2.XCO2*FGAS2.PRES;
P_O2_CA = FGAS2.XO2*FGAS2.PRES;
E0 = -46.005 + 0.01305*(FGAS1.TEMP - 1000.0);
E0 = -E0*0.0216816;
E = E0+4.308E-05*(LOG(P_H2_AN/(P_H2O_AN*P_CO2_AN))
  *FGAS1.TEMP+LOG(P_CO2_CA*P_O2_CA**0.5)*FGAS2.TEMP);
CELL VOLTAGE = E-DELTA_VOLT;
STACK VOLTAGE= CELL VOLTAGE*NO_OF_CELLS;
GROSS_POWER=STACK_VOLTAGE*CELL_CURRENT;
```

The difference in enthalpy change and gross power is saved in the variable Y, and the equation solver is called to obtain a new estimate of TTRY, the exit temperature.

```
Y=-H_DEL-GROSS_POWER;
CALL SOV(TTRY,Y,I,5.0,0.1,30,0,'MCFC');
END;
```

The power produced is saved for printout. The exit flow conditions are saved.

```
POWER.PRODUCED=GROSS_POWER;
FLC1=FGAS1;
FLC2=FGAS2;
RETURN;
```

```
MCFCOUT: ENTRY(MCFC P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT
  ('CELL TEMP=',CELL_TEMP,' K')
  (SKIP(2),COL(10),A,E(12,5),A)
  ('CELL CURRENT =',CELL_CURRENT,
   ' A') (COL(10),A,E(12,5),A)
```

```

('CELL VOLTAGE=',CELL VOLTAGE,
 ' V') (COL(10),A,F(14,2),A)
('OVERALL ISOTHERMAL HEAT OF REACTION=',HF,
 ' W') (COL(10),A,E(12,5),A)
('CELL GROSS POWER=',POWER.PRODUCED,' W')
(COL(10),A,E(12,5),A)
('NERNST POTENTIAL AT FUEL CELL EXIT=',E,' V')
(COL(10),A,E(12,5),A)
('FUEL UTILIZATION=',FUEL UTIL)
(SKIP(2),COL(10),A,E(12,5))
('OXYGEN UTILIZATION=',O2_UTIL)
(COL(10),A,E(12,5))
('CARBON DIOXIDE UTILIZATION=',CO2_UTIL)
(COL(10),A,E(12,5));
END MCFCC;

```

A.14 LIQUID-METAL DIFFUSER MODEL

A.14.1 Description of Model

The liquid-metal diffuser model (MDIF) requires one pass-through flow of the generic type LIQ. The parameters of the MDIF model are as follows:

EXIT_VELOCITY -- Specified exit flow velocity.

EFFICIENCY -- Specified efficiency of the diffuser (defined as diffuser pressure rise divided by change in velocity heads).

LENGTH -- Length of the diffuser, used in calculating pressure changes due to gravitational effects on the liquid mass. (This pressure change is added to that due to the diffuser efficiency.)

GRAV_ANGLE -- Angle that the diffuser makes with respect to the gravitational field (in degrees). At GRAV_ANGLE = 90, no gravitational effects are present.

A.14.2 Declaration Structure

```

* PROCESS NAME('MDIFC');
MDIFC: PROC( MDIF_P, LIQ_P );

DCL (MDIF_P, LIQ_P) POINTER;
DCL 1 MDIF BASED(MDIF_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,

```

```

3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 PARM,
3 EXIT_VELOCITY FLOAT(16),
3 EFFICIENCY FLOAT(16),
3 LENGTH FLOAT(16),
3 GRAV_ANGLE FLOAT(16);
DCL GP ENTRY;
DCL LIQ BASED(LIQ_P) LIKE FLC;

```

Save the inlet flow and set the specified exit velocity.

```

FLC=LIQ;
FLC.VEL =PARM.EXIT_VELOCITY;

```

On the basis of the change in velocity and the change in elevation of the diffuser's inlet and exit positions, calculate the exit enthalpy from the diffuser.

```

FLC.ENTH =LIQ.ENTH+9.80665*LENGTH*COSD(GRAV_ANGLE)
+0.5*(LIQ.VEL**2-FLC.VEL**2);

```

On the basis of the specified efficiency and any additional pressure change due to the fluid weight, calculate the exit pressure from the diffuser.

```

FLC.PRES =LIQ.PRES +0.5*LIQ.RHO/101325.*PARM.EFFICIENCY
*(LIQ.VEL**2-FLC.VEL**2)
+9.80665*LENGTH*COSD(GRAV_ANGLE)*LIQ.RHO/101325.;

```

Call the properties code, with pressure and enthalpy as inputs, to determine the other state variables of the exit flow.

```

CALL GP(NAME,FLC,10B);

```

Save the exit flow.

```

LIQ = FLC;
RETURN;
MDIFOUT :ENTRY( MDIF_P );
PUT SKIP EDIT(' ',NAME)(COL(4),A)
('Efficiency =',PARM.EFFICIENCY,
'LENGTH =',LENGTH, 'GRAV_ANGLE =',GRAV_ANGLE)
(COL(10),A,E(12,5));
END MDIFC;

```

A.15 MAGNETOHYDRODYNAMIC-GENERATOR MODEL

A.15.1 Description of Model

The magnetohydrodynamic-generator model (MG) simulates an MHD channel. The model has two entry points -- MGH, used to model the hot gas flow through the channel, and MGC, used to model the coolant flow through the channel. The MGH entry must be called before the MGC entry. Both entries require flows of the generic type GAS.

The parameters of the MG model are as follows:

AREA_INLET -- Calculated inlet flow area of the gas flow.

AREA_OUTLET -- Calculated outlet flow area of the gas flow.

B_FIELD -- Specified value of the magnetic field.

CONDUCTIVITY -- Calculated value of the electrical conductivity of the gas flow at the channel exit.

DELTA_LENGTH -- Specified length increment along the length of the channel. Calculations along the channel are performed at discrete locations, DELTA_LENGTH apart.

EXIT_PRES -- Specified value of the cutoff pressure. Calculations along the channel terminate when the calculated pressure becomes less than this value. (Actual exit pressure will not necessarily attain this specified EXIT_PRES value.)

FARADAY_CURRENT -- Calculated value of the Faraday current.

FARADAY_FIELD -- Calculated value of the Faraday electric field.

FLOW_RATIO -- Calculated ratio of the channel length to the channel height at the exit.

FRACTION_HEAT_LOSS -- Calculated ratio of the heat lost to the coolant flow to the power produced by the channel.

FRACTION_PRES_LOSS -- Calculated ratio of the pressure drop along the channel to the inlet pressure for the gas flow.

FRICTION_COEF -- Specified value of the friction coefficient (used in calculating the pressure drop along the channel).

HALL_FIELD -- Calculated value of the Hall field.

HALL_PARAMETER -- Calculated value of the maximum Hall parameter at the channel inlet or exit.

INVERTER_EFF -- Specified efficiency of the electrical inverter.

LENGTH -- Calculated length of the channel (a multiple of DELTA_LENGTH).

LOAD_FACTOR -- Specified value of the load factor along the channel.

MACH_NO_INLET -- Calculated value of the inlet-gas-flow Mach number.

MACH_NO_OUTLET -- Calculated value of the exit-gas-flow Mach number.

POWER_DENSITY -- Calculated value of the power density within the channel.

STANTON_NO -- Specified value of the Stanton number (used in calculating gas-side convective heat loss to the coolant flow).

WALL_TEMP -- Specified wall-temperature value (used in calculating heat loss to the coolant flow). This temperature is specified rather than calculated; it should be between the coolant and gas-flow temperatures.

EXTRACTED -- Calculated value of the enthalpy extracted from the gas flow.

ABSORBED -- Calculated value of the enthalpy absorbed by the coolant flow.

A.15.2 Declaration Structure

```
* PROCESS NAME('MGH');
MGH: PROC(MG_P, GAS_P);

DCL (MG_P, GAS_P, STM_P) POINTER;
DCL 1 MG_BASED(MG_P),
    2 NAME CHAR(16),
    2 FLH,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
```

```

      4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
        XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
      4 WTF FLOAT(16),
2 FLC,
      3 FNAME CHAR(16),
      3 ID CHAR(16) VARYING,
      3 ATOM(8) FLOAT(16),
      3 PROP,
      4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
      3 COMP,
      4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
        XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
      3 SOL,
      4 WTF FLOAT(16),
2 PARM,
      3 AREA_INLET FLOAT(16),
      3 AREA_OUTLET FLOAT(16),
      3 B_FIELD FLOAT(16),
      3 CONDUCTIVITY FLOAT(16),
      3 DELTA_LENGTH FLOAT(16),
      3 EXIT_PRES FLOAT(16),
      3 FARADAY_CURRENT FLOAT(16),
      3 FARADAY_FIELD FLOAT(16),
      3 FLOW_RATIO FLOAT(16),
      3 FRACTION_HEAT_LOSS FLOAT(16),
      3 FRACTION_PRES_LOSS FLOAT(16),
      3 FRICTION_COEF FLOAT(16),
      3 HALL_FIELD FLOAT(16),
      3 HALL_PARAMETER FLOAT(16),
      3 INVERTER_EFF FLOAT(16),
      3 LENGTH FLOAT(16),
      3 LOAD_FACTOR FLOAT(16),
      3 MACH_NO_INLET FLOAT(16),
      3 MACH_NO_OUTLET FLOAT(16),
      3 POWER_DENSITY FLOAT(16),
      3 STANTON_NO FLOAT(16),
      3 WALL_TEMP FLOAT(16),
      3 EXTRACTED FLOAT(16),
      3 ABSORBED FLOAT(16),
      3 PRINT FIXED BIN(15),
2 POWER,
      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
2 COST,
      3 TOTAL FLOAT(16);
DCL GAS BASED(GAS_P) LIKE FLH;
DCL STM BASED(STM_P) LIKE FLH;
DCL (SIGMA, SIGMA1, SIGMA2, BETTA, BETTA1, BETTA2, TOLD, POLD,
      HOLD, NMOLE, REJECTED) FLOAT(16),
      (DELTA_POWER, DELTA_VOLUME, AVE_TEMP) FLOAT(16),
      (PDF, PF, VB, PRES_LOSS, QC, QR, DP, DT) FLOAT(16),
      (CP, Q, S, AREA_AVE, SIDE, VOLUME, EQUIV_DIA) FLOAT(16),
      (GAMMAL INIT(1.12), GAMMA2 INIT(1.20)) FLOAT(16),
      (STEFAN_BOLTZMANN INIT(0.567E-7)) FLOAT(16),

```

```
(EGAS, EWALL INIT(0.65), PI INIT(3.14159)) FLOAT(16),
(AREA_IN, AREA_OUT, COV INIT(101325.0)) FLOAT(16),
(I,J,N) FIXED BIN(15),
(GP, GASNM) ENTRY;
```

Initialize the power to zero.

```
POWER = 0.0E0;
```

Save the inlet gas flow.

```
FLH = GAS;
```

Call the properties code to obtain consistent inlet state values and call the electrical properties code to obtain the inlet values of the conductivity and Hall parameter.

```
CALL GP(NAME,GAS,10B);
CALL SIGBET(SIGMA1,BETTA1);
```

Calculate the inlet Mach number and area and temporarily save the area in AREA_IN.

```
MACH_NO_INLET=GAS.VEL/SQRT(GAMMA1*COV*GAS.PRES/GAS.RHO);
AREA_INLET=GAS.MASS/(GAS.RHO*GAS.VEL);
AREA_IN=AREA_INLET;
```

Initialize the variables that will be incremented along the generator's channel.

```
LENGTH=0.0;
VOLUME=0.0;
FRACTION_PRES_LOSS=0.0;
CONDUCTIVITY=SIGMA1;
```

Calculate several variables, used in the equations below, that are only functions of variables that do not change along the channel.

```
VB=GAS.VEL*B_FIELD;
PF=(1.0-LOAD_FACTOR)*VB*B_FIELD;
PDF=LOAD_FACTOR*(1.0-LOAD_FACTOR)*VB**2;
```

The channel calculations are performed over an incremental length, DELTA_LENGTH, of the channel. DT and DP represent estimates of the temperature and pressure change across each incremental length. The calculations are performed twice for each length, once with the estimated temperature and pressure, and then once again as a corrective step.

```
DT=25.0*DELTA_LENGTH;
DP=0.25*DELTA_LENGTH;
```

Calculate the maximum number of incremental lengths, such that the channel is not longer than 25 m. (The actual number of incremental lengths will be determined when the pressure becomes less than EXIT_PRES.)

```
N=(25.0+0.5*DELTA_LENGTH)/DELTA_LENGTH;
```

Iterate over the number of channel lengths.

```
DO I=1 TO N WHILE(GAS.PRES > EXIT_PRES);
```

Save the temperature, enthalpy, and pressure entering the incremental length of the channel.

```
TOLD=GAS.TEMP;
HOLD=GAS.ENTH;
POLD=GAS.PRES;
```

Set the estimated exit temperature and pressure, based on DT and DP. (DT and DP will themselves be brought up to date at the end of each segment's calculations.)

```
GAS.PRES=POLD-DP;
GAS.TEMP=TOLD-DT;
```

Force the estimated temperature to be greater than the wall temperature and the pressure to be greater than the exit pressure.

```
IF GAS.PRES < EXIT_PRES THEN GAS.PRES=EXIT_PRES;
IF GAS.TEMP < WALL_TEMP THEN GAS.TEMP=WALL_TEMP;
```

Call the properties code to determine the other gas flow's state variables at the estimated conditions.

```
CALL GP(NAME,GAS,1B);
```

Perform the predictor and corrector steps.

```
DO J=1 TO 2;
```

For the corrector step, use the values of the enthalpy and pressure to determine the new state conditions of the flow out of the segment.

```
IF J=2 THEN
  CALL GP(NAME,GAS,10B);
```

Call the electrical properties code and recalculate the electrical conductivity as the square root of the segment's inlet and exit conductivity.

```
CALL SIGBET(SIGMA2,BETTA2);
SIGMA=SQRT(SIGMA1*SIGMA2);
```

Calculate the segment's power density.

```
POWER_DENSITY=SIGMA*PDF;
```

Calculate the channel geometries and hydraulic diameter.

```
AREA_OUT=GAS.MASS/(GAS.VEL*GAS.RHO);
AREA_AVE=(AREA_IN+SQRT(AREA_IN*AREA_OUT)+AREA_OUT)/3;
SIDE=SQRT(AREA_AVE);
EQUIV_DIA=AREA_AVE/(4*SIDE);
```

Calculate the pressure drop and decrement the inlet pressure.

```
PRES_LOSS=0.5*FRICTION_COEF*(GAS.MASS/AREA_AVE)*
  (GAS.VEL/EQUIV_DIA)*(DELTA_LENGTH/COV);
GAS.PRES=POLD-PF*SIGMA*DELTA_LENGTH/COV-PRES_LOSS;
```

Calculate the incremental power produced.

```
DELTA_VOLUME=DELTA_LENGTH*AREA_AVE;
DELTA_POWER=POWER_DENSITY*DELTA_VOLUME;
```

Calculate the average segment temperature and the conductive and radiant temperature transfers, QC and QR.

```
AVE_TEMP=0.5*(TOLD+GAS.TEMP);
CP=(HOLD-GAS.ENTH)/(TOLD-GAS.TEMP);
EGAS=0.5E-4*AVE_TEMP*EQUIV_DIA**0.5;
QC=GAS.MASS*STANTON_NO*CP*(AVE_TEMP-WALL_TEMP)/AREA_AVE;
QR=STEFAN_BOLTZMANN*EGAS*EWALL*(AVE_TEMP**4-WALL_TEMP**4);
```

Calculate the total enthalpy change across the segment and decrement the gas enthalpy.

```
EXTRACTED=DELTA_POWER+(QC+QR)*(4*SIDE)*DELTA_LENGTH;
GAS.ENTH=HOLD-EXTRACTED/GAS.MASS;
END;
```

Bring the values of DT and DP up to date for use in the next segment.

```
DT=TOLD-GAS.TEMP;
DP=POLD-GAS.PRES;
SIGMA1=SIGMA2;
```

Bring the segment inlet area up to date and increment the other variables integrated along the channel.

```
AREA_IN=AREA_OUT;
LENGTH=LENGTH+DELTA_LENGTH;
VOLUME=VOLUME+DELTA_VOLUME;
POWER.PRODUCED=POWER.PRODUCED+DELTA_POWER;
FRACTION_PRES_LOSS=FRACTION_PRES_LOSS+PRES_LOSS;
END;
```

Set the channel output variables and bring the power variables up to date.

```

CONDUCTIVITY=SQRT(CONDUCTIVITY*SIGMA2);
AREA_OUTLET=AREA_OUT;
EXTRACTED=(FLH.ENTH-GAS.ENTH)*GAS.MASS;
MACH_NO_OUTLET=GAS.VEL/SQRT(GAMMA2*COV*GAS.PRES/GAS.RHO);
HALL_PARAMETER=MAX(BETTA1,BETTA2);
ABSORBED=EXTRACTED-POWER.PRODUCED;
REJECTED=POWER.PRODUCED*(1.0-INVERTER_EFF);
POWER.PRODUCED=POWER.PRODUCED-REJECTED;
FRACTION_HEAT_LOSS=ABSORBED/POWER.PRODUCED;
FRACTION_PRES_LOSS=FRACTION_PRES_LOSS/FLH.PRES;
FLOW_RATIO=LENGTH/SIDE;
BETTA=SQRT(BETTA1*BETTA2);
FARADAY_FIELD=LOAD_FACTOR*GAS.VEL*B_FIELD;
FARADAY_CURRENT=CONDUCTIVITY*(GAS.VEL*B_FIELD-FARADAY_FIELD);
HALL_FIELD=(BETTA/CONDUCTIVITY)*FARADAY_CURRENT;
COST.TOTAL = 0.0;

```

Save the exit gas flow from the channel.

```

FLH = GAS;
RETURN;

```

```

MGC: ENTRY(MG_P,STM_P);

```

Using the energy absorbed, as calculated within the gas-side (hot) entry, calculate the exit enthalpy of the coolant side and call the properties code to obtain the other state variables of the flow.

```

STM.ENTH=STM.ENTH+ABSORBED/STM.MASS;
CALL GP(NAME,STM,10B);

```

Save the coolant flow.

```

FLC=STM;
RETURN;
SIGBET: PROC(SIGMA,BETTA);
DCL (SIGMA,BETTA,T1,T2,T3,SUM, NMOLE, ELECTRON_CONCENTRATION,
ELECTRON_THERMAL_VELOCITY, POTASSIUM_CONCENTRATION, AVOGADRO
INIT(6.0228E23), BOLTZMANN INIT(1.38047E-23), ELECTRON_CHARGE
INIT(1.60203E-19), ELECTRON_MASS INIT(9.11E-31), PERMITTIVITY
INIT(8.85525E-12), PLANK INIT(6.6242E-34),
POTASSIUM_IONIZATION_POTENTIAL INIT(4.34)) FLOAT(16);
ELECTRON_THERMAL_VELOCITY=
SQRT(8.0*GAS.TEMP/PI*(BOLTZMANN/ELECTRON_MASS));
CALL GASNM(GAS.COMP,NMOLE);
POTASSIUM_CONCENTRATION=1000*AVOGADRO*GAS.RHO*NMOLE*GAS.XK;
SUM=1000*AVOGADRO*1E-20*GAS.RHO*NMOLE*(
GAS.XK*400+GAS.XH2*13.8+GAS.XO2*3+GAS.XO*20+GAS.XN2*6.5+
GAS.XH2O*75+GAS.XCO2*15+GAS.XCO*8);
ELECTRON_CONCENTRATION=SQRT((POTASSIUM_CONCENTRATION/PLANK)*
((2*PI*ELECTRON_MASS*BOLTZMANN*GAS.TEMP)**1.5/PLANK)*

```

```

      (EXP(-(ELECTRON_CHARGE*POTASSIUM_IONIZATION_POTENTIAL)/
      (BOLTZMANN*GAS.TEMP)))/PLANK));
IF ELECTRON_CONCENTRATION=0.0 THEN
  DO;
    SIGMA,BETTA=0.0;
    RETURN;
  END;
T1=(ELECTRON_CONCENTRATION*ELECTRON_CHARGE*ELECTRON_CHARGE)/
  (ELECTRON_MASS*ELECTRON_THERMAL_VELOCITY);
T2=((ELECTRON_CHARGE/BOLTZMANN)*(
  ELECTRON_CHARGE/(8*PI*PERMITTIVITY*GAS.TEMP)))*2;
T3=12*PI*(PERMITTIVITY*BOLTZMANN*GAS.TEMP/(
  ELECTRON_CHARGE**2))*1.5/ELECTRON_CONCENTRATION**0.5;
SIGMA=T1/√SUM+3.9*ELECTRON_CONCENTRATION*T2*LOG(T3));
BETTA=(SIGMA*B_FIELD)/(ELECTRON_CONCENTRATION*ELECTRON_CHARGE);
END SIGBET;

MGOUT: ENTRY(MG_P);
  PUT SKIP EDIT(' ',NAME)(COL(4),A);
  PUT SKIP(2) EDIT(
    'STANTON NO. = ',STANTON_NO,
    'FRICTION COEFFICIENT = ',FRICTION_COEF,
    'EXIT PRESSURE = ',EXIT_PRES,
    'WALL TEMPERATURE = ',WALL_TEMP,
    'LOAD FACTOR = ',LOAD_FACTOR,
    'FARADAY FIELD = ',FARADAY_FIELD,
    'FARADAY CURRENT = ',FARADAY_CURRENT,
    'HALL FIELD = ',HALL_FIELD,
    'MAGNETIC FIELD INTENSITY = ',B_FIELD,
    'PERCENTAGE HEAT LOSS = ',100*FRACTION_HEAT_LOSS,
    'PERCENTAGE PRESSURE LOSS = ',100*FRACTION_PRES_LOSS,
    'MAXIMUM HALL PARAMETER = ',HALL_PARAMETER,
    'AVERAGE CONDUCTIVITY = ',CONDUCTIVITY,
    'INLET MACH NO. = ',MACH_NO_INLET,
    'OUTLET MACH NO. = ',MACH_NO_OUTLET,
    'POWER DENSITY = ',POWER_DENSITY,
    'FLOW RATIO (L/D) = ',FLOW_RATIO,
    'INLET AREA = ',AREA_INLET,
    'OUTLET AREA = ',AREA_OUTLET,
    'CHANNEL LENGTH = ',LENGTH)
    (COL(10),A,E(12,5));
END MGH;

```

A.16 LIQUID-METAL MAGNETOHYDRODYNAMIC-GENERATOR MODEL

A.16.1 Description of Model

The two-component liquid-metal MHD-generator model (MMHD) requires two pass-through flows of the generic types GAS and LIQ. The first flow represents the predominant gaseous component of the two-component flow, while the second represents the liquid component.

The parameters of the MMHD model are as follows:

EFFICIENCY -- Specified isentropic expansion efficiency of the generator.

EXIT_PRES -- Specified exit pressure from the generator.

SLIP_RATIO -- Specified ratio of the gas velocity to the liquid velocity at the exit of the generator.

TEMP_DIFF -- Specified difference between the liquid temperature and that of the gas at the exit of the generator.

LENGTH -- Length of the generator (used in calculating the gain or loss in energy due to the effects of gravity on the mass of fluid within the generator).

GRAV_ANGLE -- Specified angle the generator makes with the gravitational field.

VOID_FRACTION -- Calculated void fraction at the exit of the generator.

A.16.2 Declaration Structure

```
* PROCESS NAME('MMHDC');
MMHDC: PROC(MMHD_P, GAS_P, LIQ_P);

DCL (MMHD_P, GAS_P, LIQ_P) POINTER;
DCL 1 MMHD BASED(MMHD_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
```

```

      4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
        XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
      3 SOL,
      4 WTF FLOAT(16),
      2 PARM,
      3 EFFICIENCY FLOAT(16),
      3 EXIT PRES FLOAT(16),
      3 SLIP_RATIO FLOAT(16),
      3 TEMP_DIFF FLOAT(16),
      3 LENGTH FLOAT(16),
      3 GRAV_ANGLE FLOAT(16),
      3 VOID_FRACTION FLOAT(16),
      2 POWER,
      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL (S_TOTAL,H_EXIT,H_AVAILABLE,RESULT) FLOAT(16);
DCL SOV ENTRY( FLOAT(16), FLOAT(16), FIXED BIN(15),
  FLOAT(16), FLOAT(16), FIXED BIN(15), FIXED BIN(15), CHAR(8));
DCL GP ENTRY;
DCL I FIXED BIN(15);
DCL GAS BASED(GAS_P) LIKE FLC1;
DCL LIQ BASED(LIQ_P) LIKE FLC2;

```

Initialize the power to zero and save the inlet flows.

```

POWER=0.0;
FLC1=GAS;
FLC2=LIQ;

```

Set the exit pressure to the specified value.

```

GAS.PRES =PARM.EXIT_PRES;
LIQ.PRES =PARM.EXIT_PRES;

```

The liquid velocity is assumed to be the same as the inlet value; the exit gas velocity is then determined using the specified slip ratio.

```

GAS.VEL =LIQ.VEL*PARM.SLIP_RATIO;

```

The inlet value of the total entropy of the gas and liquid flows is calculated and saved in S_TOTAL.

```

S_TOTAL =GAS.MASS*GAS.ENTP+LIQ.MASS*LIQ.ENTP;

```

The isentropic expansion to the exit pressure is calculated by iterating over the gas enthalpy until the calculated enthalpy of the gas-liquid mixture is equal to its inlet value. At each iteration, the gas enthalpy is used in the properties code to evaluate its temperature and entropy; the liquid temperature is obtained using the specified liquid-gas temperature difference. The properties code is called with the liquid to determine its entropy, and the total entropy of the gas and liquid can then be found. (This method will be valid so long as the liquid does not vaporize (i.e., its properties must be able to be determined from its temperature and pressure).

```

DO I=1 TO 21;
  CALL GP(NAME,GAS,10B);
  LIQ.TEMP=GAS.TEMP+PARM.TEMP_DIFF;
  CALL GP(NAME,LIQ,1B);
  RESULT=GAS.MASS*GAS.ENTP+LIQ.MASS*LIQ.ENTP-S_TOTAL;
  CALL SOV(GAS.ENTH,RESULT,I,-1E3,1E-3,21,0,'MMHD1');
END;

```

At this point, the variables GAS.ENTH and LIQ.ENTH will contain their values at the isentropic expansion. Thus, the total enthalpy available to generate power can be determined by subtraction of the mixture enthalpy from its inlet value. Using the specified efficiency, the exit enthalpy of the mixture is obtained.

```

H_AVAILABLE =GAS.MASS*(FLC1.ENTH-GAS.ENTH)
              +LIQ.MASS*(FLC2.ENTH-LIQ.ENTH);
H_EXIT =FLC1.MASS*FLC1.ENTH+FLC2.MASS*FLC2.ENTH
        -PARM.EFFICIENCY*H_AVAILABLE;

```

The exit values of the individual gas and liquid flows can be obtained using a method similar to that employed in obtaining the isentropic state point. An iteration is performed over the gas enthalpy. With a given value of the gas enthalpy, the gas temperature can be obtained, followed by the liquid temperature, the liquid enthalpy, and the mixture enthalpy. (This method will be valid only if the liquid properties can be obtained as a function of temperature and pressure.)

```

DO I=1 TO 21;
  CALL GP(NAME,GAS,10B);
  LIQ.TEMP=GAS.TEMP+PARM.TEMP_DIFF;
  CALL GP(NAME,LIQ,1B);
  RESULT=GAS.MASS*GAS.ENTH+LIQ.MASS*LIQ.ENTH-H_EXIT;
  CALL SOV(GAS.ENTH,RESULT,I,1E3,1E0,21,0,'MMHD2');
END;

```

One can then find the total power produced for the expansion process, as well as for any changes in velocity or potential energy across the gravitational field.

```

POWER.PRODUCED =PARM.EFFICIENCY*H_AVAILABLE
                +(GAS.MASS+LIQ.MASS)*9.80665*LENGTH*COSD(GRAV_ANGLE)
                -FLC1.MASS*0.5*(FLC1.VEL**2-GAS.VEL**2);

```

The exit void fraction is calculated for printout.

```

VOID_FRACTION=GAS.MASS/(GAS.MASS+LIQ.MASS*GAS.RHO*GAS.VEL/
                        LIQ.RHO/LIQ.VEL);

```

The exit flows are saved.

```

FLC1=GAS;
FLC2=LIQ;
RETURN;
MMHDOUT: ENTRY( MMHD_P );

```

```

PUT SKIP EDIT(' ',NAME)(COL(4),A)
('Efficiency =' ,PARAM.EFFICIENCY,
'Power =' ,POWER.PRODUCED, 'VOID FRAC. =' ,VOID_FRACTION,
'LENGTH =' ,LENGTH, 'GRAV ANGLE =' ,GRAV_ANGLE)
(COL(10),A,E(12,5));
END MMHDC;

```

A.17 LIQUID-METAL NOZZLE MODEL

A.17.1 Description of Model

The liquid-metal nozzle model (MNOZ) requires one pass-through flow of the generic type LIQ. The parameters of the MNOZ model are as follows:

EFFICIENCY -- Specified efficiency of the nozzle, defined as the change in velocity head divided by the change in pressure across the nozzle.

EXIT_VELOCITY -- Specified exit velocity from the nozzle.

LENGTH -- Specified length of the nozzle (used in determining gravitational effects on the pressure changes across the nozzle due to the mass of fluid within the nozzle).

GRAV_ANGLE -- Specified angle that the nozzle makes with the gravitational field.

A.17.2 Declaration Structure

```

* PROCESS NAME('MNOZC');
MNOZC: PROCEDURE ( MNOZ_P, LIQ_P );

DCL (MNOZ_P, LIQ_P) POINTER;
DCL 1 MNOZ BASED(MNOZ_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,

```

```

      3 EFFICIENCY FLOAT(16),
      3 EXIT_VELOCITY FLOAT(16),
      3 LENGTH FLOAT(16),
      3 GRAV_ANGLE FLOAT(16);
DCL LIQ BASED(LIQ_P) LIKE FLC;
DCL GP ENTRY;

```

Save the inlet flow and assign the specified exit velocity.

```

FLC=LIQ;
FLC.VEL =PARAM.EXIT_VELOCITY;

```

On the basis of the change in velocity and any change in elevation between the inlet and exit of the nozzle, adjust the exit enthalpy of the flow.

```

FLC.ENTH =LIQ.ENTH+9.80665*LENGTH*COSD(GRAV_ANGLE)
+0.5*(LIQ.VEL**2-FLC.VEL**2);

```

Calculate the exit pressure of the flow, based on the specified efficiency and any additional pressure due to the elevation change.

```

FLC.PRES=LIQ.PRES-0.5*FLC.RHO*(FLC.VEL**2-LIQ.VEL**2)
/(PARAM.EFFICIENCY*101325.0)
+9.80665*LENGTH*COSD(GRAV_ANGLE)*FLC.RHO/101325.;

```

Call the properties code, with enthalpy and pressure as inputs, to determine the other state variables of the flow.

```

CALL GP(NAME,FLC,10B);

```

Save the flow.

```

LIQ = FLC;
RETURN;
MNOZOUT :ENTRY( MNOZ P );
PUT SKIP EDIT(' ',NAME)(COL(4),A)
('EFFICIENCY =',PARAM.EFFICIENCY,'EXIT VELOCITY=',EXIT_VELOCITY,
'LENGTH =',LENGTH, 'GRAV_ANGLE =',GRAV_ANGLE)
(COL(10),A,E(12,5));
END MNOZC;

```

A.18 LIQUID-METAL PIPE MODEL

A.18.1 Description of Model

Liquid-metal flow through a pipe is modeled by MPIP, which requires one pass-through flow of the generic type LIQ. The parameters of the MPIP model are as follows:

FRIC_FAC -- Calculated friction factor of the flow within the pipe, established using a simple Reynolds-number correlation.

VISC -- Specified viscosity of the flow within the pipe.

RE -- Calculated Reynolds number of the flow within the pipe.

FD -- Calculated pressure drop per unit length of pipe due to the frictional effects of the flow on the pipe.

FG -- Calculated pressure changes per unit length of pipe due to the effects of gravity on the mass of fluid within the pipe.

AREA -- Calculated flow area of the pipe, based on the inlet mass flow rate, density, and velocity.

DIAMETER -- Calculated pipe diameter, based on the calculated area.

LENGTH -- Specified length of the pipe.

GRAV_ANGLE -- Specified angle the pipe makes with the gravitational field.

A.18.2 Declaration Structure

```
* PROCESS NAME('MPIP');
MPIP: PROC(MPIP_P,LIQ_P);

DCL (MPIP_P,LIQ_P) POINTER;
DCL 1 MPIP_BASED(MPIP_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
```

```

      4 WTF FLOAT(16),
2 PARM,
      3 (FRIC_FACT,
        VISC,
        RE,
        FD,
        FG,
        AREA,
        DIAMETER,
        LENGTH,
        GRAV_ANGLE) FLOAT(16);
DCL (GP,VISSOLN) ENTRY;
DCL LIQ BASED(LIQ_P) LIKE FLC;

```

Calculate the area and diameter of the pipe.

```

AREA=LIQ.MASS/(LIQ.RHO*LIQ.VEL);
DIAMETER=SQRT(4.0*AREA/3.14159);

```

Call the liquid-metal viscosity procedure.

```

CALL VISSOLN(LIQ.ID,LIQ.TEMP,VISC);

```

With the viscosity known, calculate the Reynolds number, the friction factor (a simple laminar-flow relation is used), and the pressure drop per unit length due to frictional drag.

```

RE=LIQ.RHO*LIQ.VEL*DIAMETER/VISC;
FRIC_FACT=0.3164/RE**0.25;
FD=FRIC_FACT*LIQ.RHO*LIQ.VEL**2/(2.0*DIAMETER);

```

Calculate the pressure change per unit length due to the gravitational head.

```

FG=LIQ.RHO*9.80665*COSD(GRAV_ANGLE);

```

Adjust the flow's enthalpy and pressure due to the pressure changes and gravitational fields.

```

LIQ.ENTH=LIQ.ENTH+FG*LENGTH/LIQ.RHO;
LIQ.PRES=LIQ.PRES-(FD-FG)*LENGTH/101325.;

```

Call the properties code, with the pressure and enthalpy as inputs, to obtain the other variables of the flow.

```

CALL GP(NAME,LIQ,10B);

```

Save the exit flow. (Technically, the exit velocity of the flow will have changed due to the pressure drop within the constant-area pipe; this change will be small, and it is neglected in this model.)

```

FLC=LIQ;
RETURN;
MPIPOUT: ENTRY(MPIP P);
PUT SKIP EDIT(' ',NAME)(COL(4),A)
('LENGTH=',LENGTH,'GRAV ANGLE=',GRAV_ANGLE,'AREA=',AREA,
'DIAMETER=',DIAMETER,'RE=',RE,'FRIC FAC=',FRIC_FACT,
'FD=',FD,'FG=',FG)
(COL(10),A,E(12,5));
END MPIPC;

```

A.19 FLOW-MIXER MODEL

A.19.1 Description of Model

The flow-mixer model (MX) requires two flows. The first is a pass-through flow, representing one of the two input flows and also the output flow. The second flow is the second input flow. Both of these flows are of the generic type GAS.

The model does not require any parameters.

A.19.2 Declaration Structure

```

* PROCESS NAME('MXC');
MXC: PROC(MX_P,F1_P,F2_P);

DCL (MX_P,F1_P,F2_P) POINTER;
DCL 1 MX_ BASED(MX_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (T,P,H,S,Q,R,V,M) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (T,P,H,S,Q,R,V,M) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,

```

```

      4 WTF FLOAT(16);
DCL F1 BASED(F1_P) LIKE FLC1,
    F2 BASED(F2_P) LIKE FLC2,
    A PTR POINTER,
    ARRY(23) FLOAT(16) BASED(A_PTR),
    (K E,N1,N2) FLOAT(16),
    (GP,GASBW,GASNM) ENTRY;

```

The inlet flows are saved in FLC1 and FLC2.

```

FLC1=F1;
FLC2=F2;

```

If the first flow is a "GAS" and the second is "H2O," then the two flows are not using the same thermodynamic property calculations. An approximation is made by redefining the second flow's enthalpy, using its temperature and assuming that it is all in the superheated region before being mixed with the gas. (This is only an approximation; if GAS and H2O flows must be mixed, the result should be used with caution.)

```

IF F1.ID='GAS' & F2.ID='H2O' THEN
DO;
  F2.ID='GAS';
  F2.COMP.XH2O=1.0;
  CALL GASBW(F2.COMP,F2.ATOM);
  CALL GP(NAME,F2,1B);
END;

```

A similar readjustment of the flow enthalpy is made if a GAS flow that is entirely water is mixed with an H2O flow. (This combination should be used with caution.)

```

IF F1.ID='H2O' & F2.ID='GAS' THEN
DO;
  F2.ID='H2O';
  CALL GP(NAME,F2,1B);
END;

```

When both flows are of type GAS, then the two flows are mixed by combining their species' molar flow rates.

```

IF F1.ID='GAS' & F2.ID='GAS' THEN
DO;

```

Call GASNM to determine the total number of moles per kilogram of flow.

```

CALL GASNM(F1.COMP,N1);
CALL GASNM(F2.COMP,N2);

```

Redefine, temporarily, the species' molar-fraction structure to represent the total species' molar flow rates.

```

F1.COMP=N1*F1.COMP*F1.M*(1.0-F1.WTF);
F2.COMP=N2*F2.COMP*F2.M*(1.0-F2.WTF);

```

Combine the molar flow rates of the two streams.

```
F1.COMP=F1.COMP+F2.COMP;
```

Using an overlay on a dummy array of the F1.COMP, sum the total number of moles for the flow.

```
A_PTR=ADDR(F1.COMP);  
N1=SUM(ARRAY);
```

Calculate the molar fractions of the flow and call GASBW to obtain the new ATOM array for the flow.

```
IF N1>0.0 THEN  
  F1.COMP=F1.COMP/N1;  
ELSE  
  F1.COMP=0.0;  
CALL GASBW(F1.COMP,F1.ATOM);
```

Redefine the entrained-solid weight fraction.

```
F1.WTF=(F1.WTF*F1.M+F2.WTF*F2.M)/(F1.M+F2.M);  
END;
```

Sum the total mass flow for the two flows.

```
F1.M=F1.M+F2.M;
```

Set the exit pressure of the mixed flows equal to the minimum inlet pressure. (This method of defining the exit pressure may cause the pressure to oscillate between two values in an uncontrollable manner during iterative loops.)

```
F1.P=MIN(F1.P,F2.P);
```

Calculate the total kinetic energy of the two flows. Define the exit velocity of the combined flows as the velocity that would give this same kinetic energy.

```
K_E=0.5*(FLC1.M*FLC1.V*FLC1.V+FLC2.M*FLC2.V*FLC2.V);  
F1.V=SQRT(2.0*K_E/F1.M);
```

Add the total enthalpy of the two flows together and call the properties code to obtain the new exit flow conditions.

```
F1.H=(FLC1.M*F1.H+FLC2.M*F2.H)/F1.M;  
CALL GP(NAME,F1,10B);
```

Save the exit flow.

```
FLC1=F1;  
RETURN;
```

```
MXOUT: ENTRY(MX_P);
END MXC;
```

A.20 GAS-NOZZLE MODEL

A.20.1 Description of Model

The gas-nozzle model (NZ) requires one pass-through flow of the generic type GAS. The parameters of the NZ model are as follows:

EFFICIENCY -- Specified efficiency of the nozzle expansion, defined as the isentropic pressure drop necessary to accelerate the flow to the specified exit velocity, divided by the actual pressure drop.

EXIT_VEL -- Specified exit velocity of the gas flow.

PRINT -- Specified print switch; if set to a number greater than zero, this switch will print out the iterations within the model used in obtaining the isentropic exit pressure.

A.20.2 Declaration Structure

```
* PROCESS NAME('NZC');
NZC: PROC(NZ_P, GAS_P);

DCL (NZ_P, GAS_P) POINTER;
DCL 1 NZ BASED(NZ_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 EFFICIENCY FLOAT(16),
    3 EXIT_VEL FLOAT(16),
    3 PRINT FIXED BIN(15),
    2 COST,
    3 TOTAL FLOAT(16);
DCL GAS BASED(GAS_P) LIKE FLC,
    (Q, R, S, DIFFERENCE) FLOAT(16),
    I FIXED BIN(15),
```

```

SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16)
          ,FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*)),
GP ENTRY;

```

Save the inlet flow and then calculate the flow's exit enthalpy using conservation of energy and the specified exit velocity.

```

FLC = GAS;
FLC.ENTH=GAS.ENTH+0.5*GAS.VEL**2-0.5*EXIT_VEL**2;

```

Iterate over the exit pressure, keeping the flow's entropy equal to its inlet value until the flow's enthalpy is equal to this new exit enthalpy value. Thus, the isentropic pressure is obtained.

```

DO I=1 TO 15;
  CALL GP(NAME,GAS,11B);
  DIFFERENCE=FLC.ENTH-GAS.ENTH;
  CALL SOV(GAS.PRES,DIFFERENCE,I,1.0,20.0,15,PRINT,'NZ');
END;

```

Set the gas flow's exit velocity.

```

GAS.VEL=EXIT_VEL;

```

Using the specified efficiency and the just-calculated isentropic pressure, calculate the actual exit pressure.

```

GAS.PRES=FLC.PRES+(GAS.PRES-FLC.PRES)/EFFICIENCY;

```

Call the properties code to determine the other state variables of the flow.

```

CALL GP(NAME,GAS,10B);
COST.TOTAL = 0.0;

```

Save the exit flow.

```

FLC = GAS;
RETURN;

NZOUT: ENTRY(NZ_P);
  PUT SKIP EDIT(' ',NAME)(COL(4),A);
  PUT SKIP(2) EDIT(
    'EFFICIENCY = ',EFFICIENCY,
    'EXIT VELOCITY = ',EXIT_VEL)
    (COL(10),A,E(12,5));
  END NZC;

```

A.21 PHOSPHORIC ACID FUEL-CELL MODEL

A.21.1 Description of Model

The phosphoric acid fuel-cell model (PAFC) requires two flows, both of the generic type GAS. The first flow represents the anode flow; the second, the cathode flow.

The parameters of the PAFC model are as follows:

CELL_CURRENT -- Specified current through each cell.

CELL_VOLTAGE -- Calculated cell voltage.

CELL_TEMP -- Specified average temperature of cell.

STACK_VOLTAGE -- Calculated total voltage across cell stack.

NO_OF_CELLS -- Specified total number of cells in stack.

DELTA_VOLT -- Specified difference between Nernst potential at fuel-cell exit and cell voltage.

FUEL_UTIL -- Calculated value of fuel utilization.

O2_UTIL -- Calculated value of oxygen utilization.

HF -- Calculated value of overall isothermal heat of reaction.

E -- Calculated Nernst potential at fuel-cell exit.

A.21.2 Declaration Structure

```
* PROCESS NAME('PAFCC');
PAFCC: PROC(PAFC_P,FGAS1_P,FGAS2_P);

DCL (PAFC_P,FGAS1_P,FGAS2_P) POINTER;
DCL 1 PAFCC BASED(PAFC_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
```

```

3 SOL,
  4 WTF FLOAT(16),
2 FLC2,
  3 FNAME CHAR(16),
  3 ID CHAR(16) VARYING,
  3 ATOM(8) FLOAT(16),
  3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
  3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
  3 SOL,
    4 WTF FLOAT(16),
2 PARM,
  3 CELL_CURRENT FLOAT(16),
  3 CELL_VOLTAGE FLOAT(16),
  3 STACK_VOLTAGE FLOAT(16),
  3 CELL_TEMP FLOAT(16),
  3 NO_OF_CELLS FLOAT(16),
  3 DELTA_VOLT FLOAT(16),
  3 FUEL_UTIL FLOAT(16),
  3 O2_UTIL FLOAT(16),
  3 (HF,E) FLOAT(16),
2 POWER,
  3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL FGAS1 BASED(FGAS1_P) LIKE FLC1, FGAS2 BASED(FGAS2_P) LIKE FLC2;
DCL (Y,O2_MOL,H2_MOL,H2O_MOL,F,H_DEL_TC,MOL_WT,
  H_DEL_TTRY,HAIN,HAIN_TC,HAOUT,HAOUT_TC,
  EO,P_H2O_CA,P_H2_AN,P_O2_CA,GROSS_POWER,
  HGIN,HGIN_TC,HCOUT,HCOUT_TC,QT) FLOAT(16);
DCL (GASBW,GASMR,GP) ENTRY;
DCL SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
  FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*));

```

Perform the anode calculations: Set Faraday's constant and initialize the power to zero.

```

F=96487.0;
POWER=0.0;

```

Save the value of the inlet anode flow enthalpy.

```
HAIN = FGAS1.ENTH*FGAS1.MASS;
```

Set the anode flow temperature to the specified cell temperature. Call the properties code to obtain the state conditions of the flow at this temperature.

```

FGAS1.TEMP = CELL_TEMP;
CALL GP(NAME,FGAS1,1B);

```

Save the value of the anode flow enthalpy at this cell temperature.

```
HAIN_TC = FGAS1.ENTH*FGAS1.MASS;
```

Since the fuel cell only works with the gas flow streams, any entrained-solid flow is temporarily subtracted and saved in the FLC1 flow variables.

```
FLC1.WTF=FGAS1.WTF*FGAS1.MASS;
FLC1.MASS=FGAS1.MASS-FLC1.WTF;
```

The procedure GASMR will calculate the molar flow rates of the flow, given the mass flow rate and the species' molar fractions, if its first argument is 1. These molar flow rates are temporarily stored in the FLC1 variables.

```
CALL GASMR(1B,FGAS1.COMP,FLC1.MASS,FLC1.COMP,MOL_WT);
```

Calculate the hydrogen concentration in the anode flow.

```
H2_MOL = CELL_CURRENT*NO_OF_CELLS/(2000.0*F);
```

Check whether or not there is sufficient hydrogen concentration in the anode flow. If not, indicate how much more is needed.

```
IF (H2_MOL > FLC1.XH2) THEN
DO;
  PUT SKIP(2) EDIT('** ERROR IN PAFC:',
    'NOT ENOUGH FUEL IN ANODE INLET STREAM.',
    'INCREASE H2 MOLE RATE TO ',H2_MOL*1.01,
    '(MOL/S) OR MORE.')
    (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,
    E(13,5),X(2),A);
  STOP;
END;
```

Calculate the fuel utilization.

```
FUEL_UTIL = H2_MOL/FLC1.XH2;
```

Calculate the moles of H₂O and O₂ that cross the cell and adjust the H₂ concentration at the anode flow.

```
H2O_MOL = H2_MOL;
O2_MOL = H2_MOL/2.0;
FLC1.XH2 = FLC1.XH2 - H2_MOL;
```

Recall the GASMR procedure, with its first argument set to 2, to convert the molar flow rates back to species' molar fractions and to obtain the total mass flow rate. These values are placed into the FGAS1 flow variables.

```
CALL GASMR(10B,FGAS1.COMP,FGAS1.MASS,FLC1.COMP,MOL_WT);
```

Given the new values of the FGAS1 composition, call GASBW to obtain the ATOM array.

```
CALL GASBW(FGAS1.COMP,FGAS1.ATOM);
```

Add any entrained solids back to the flow and call the properties code to obtain the exit flow conditions.

```
FGAS1.MASS=FGAS1.MASS+FLC1.WTF;
FGAS1.WTF=FLC1.WTF/FGAS1.MASS;
CALL GP(NAME,FGAS1,1B);
```

Save the exit flow enthalpy from the anode.

```
HAOUT_TC = FGAS1.ENTH*FGAS1.MASS;
```

Next, perform the cathode calculations: Save the cathode inlet enthalpy.

```
HCIN = FGAS2.ENTH*FGAS2.MASS;
```

Set the cathode flow temperature to equal the cell temperature.

```
FGAS2.TEMP = CELL_TEMP;
```

Call the properties code to obtain the state conditions of the cathode flow at the cell temperature and save the value of its enthalpy.

```
CALL GP(NAME,FGAS2,1B);
HCIN_TC = FGAS2.ENTH*FGAS2.MASS;
```

Subtract any entrained solids and call GASMR to obtain the molar flow rates, as was done in the case of the anode stream.

```
FLC2.WTF=FGAS2.WTF*FGAS2.MASS;
FLC2.MASS=FGAS2.MASS-FLC2.WTF;
CALL GASMR(1B,FGAS2.COMP,FLC2.MASS,FLC2.COMP,MOL_WT);
```

Adjust the species' molar rates to reflect the crossover of H_2O and O_2 .

```
O2_UTIL = O2_MOL/FLC2.XO2;
FLC2.XH2O = FLC2.XH2O+H2O_MOL;
FLC2.XO2 = FLC2.XO2-O2_UTIL;
```

Check whether or not there is sufficient O_2 with the stream. If not, indicate how much would be needed and terminate.

```
IF (FLC2.XO2 < 0.0) THEN
DO;
  PUT SKIP(2) EDIT('** ERROR IN PAFC:',
    'NOT ENOUGH O2 IN CATHODE INLET STREAM.',
    'INCREASE O2 MOLE RATE TO ',1.25*O2_UTIL,
    '(MOL/S) OR MORE')
    (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,E(13,5),
    X(2),A);
  STOP;
END;
```

Reconvert back to molar fractions, set the new ATOM array values, add the subtracted entrained solids, and call the properties code.

```
CALL GASMR(10B,FGAS2.COMP,FGAS2.MASS,FLC2.COMP,MOL_WT);
CALL GASBW(FGAS2.COMP,FGAS2.ATOM);
FGAS2.MASS=FGAS2.MASS+FLC2.WTF;
FGAS2.WTF=FLC2.WTF/FGAS2.MASS;
CALL GP(NAME,FGAS2,1B);
```

Save the exit cathode flow enthalpy.

```
HCOUT_TC = FGAS2.ENTH*FGAS2.MASS;
```

Finally, perform the energy balance calculations: Save the total enthalpy change across the cell (at the cell temperature).

```
H_DEL_TC=HAOUT_TC-HAIN_TC+HCOUT_TC-HCIN_TC;
HF=-H_DEL_TC;
```

The actual exit temperature of the cell will be determined by varying the exit temperature until the total enthalpy change across the cell stack represents the gross power produced by the cell. For the first approximation, the exit temperature is taken to equal the cell temperature. Thereafter, the exit temperature is controlled by the one-dimensional equation solver, SOV.

```
TTRY=CELL_TEMP;
DO I=1 TO 25;
```

Set the anode exit temperature and call the properties code to obtain the exit enthalpy.

```
FGAS1.TEMP = TTRY;
CALL GP(NAME,FGAS1,1B);
HAOUT = FGAS1.ENTH*FGAS1.MASS;
```

Set the cathode exit temperature and call the properties code to obtain the exit enthalpy.

```
FGAS2.TEMP = TTRY;
CALL GP(NAME,FGAS2,1B);
HCOUT = FGAS2.ENTH*FGAS2.MASS;
```

Calculate the total enthalpy change across the cell.

```
H_DEL=HAOUT-HAIN+HCOUT-HCIN;
```

The gross power output from the cell depends on the stack voltage, and hence, the cell voltage. The cell voltage is taken as the Nernst potential minus some specified voltage drop, DELTA_VOLT. The Nernst potential is a function of the partial pressures of the species at the exits of the anode and cathode streams. These partial pressures are calculated first, followed by the Nernst potential, the cell voltage, the stack voltage, and the gross power output.

```

P_H2 AN = FGAS1.XH2*FGAS1.PRES;
P_H2O CA = FGAS2.XH2O*FGAS2.PRES;
P_O2 CA = FGAS2.XO2*FGAS2.PRES;
E0 = -51.147 + 0.0118984*(FGAS1.TEMP - 600.0);
E0 = -E0*0.0216816;
E = E0+4.308E-05*(LOG(P_H2 AN)*FGAS1.TEMP +
  LOG(P_O2 CA**0.5/P_H2O CA)*FGAS2.TEMP);
CELL VOLTAGE = E-DELTA VOLT;
STACK VOLTAGE= CELL VOLTAGE*NO_OF_CELLS;
GROSS_POWER=STACK VOLTAGE*CELL_CURRENT;

```

The difference in enthalpy change and gross power is saved in the variable Y. The equation solver is called to obtain a new estimate of TTRY, the exit temperature.

```

Y=-H_DEL-GROSS_POWER;
CALL SOV(TTRY,Y,I,5.0,0.1,25,0,'PAFC_');
END;

```

The power produced is saved for printout; the exit flow conditions are saved.

```

POWER.PRODUCED=GROSS_POWER;
FLC1=FGAS1;
FLC2=FGAS2;
RETURN;

PAFCOUT: ENTRY(PAFC P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT
  ('CELL TEMP=',CELL_TEMP,' K')
  (SKIP(2),COL(10),A,E(12,5),A)
  ('CELL CURRENT =',CELL_CURRENT,
   ' A') (COL(10),A,E(12,5),A)
  ('CELL VOLTAGE=',CELL_VOLTAGE,
   ' V') (COL(10),A,F(14,2),A)
  ('OVERALL ISOTHERMAL HEAT OF REACTION=',HF,
   ' W') (COL(10),A,E(12,5),A)
  ('CELL GROSS POWER=',POWER.PRODUCED,' W')
  (COL(10),A,E(12,5),A)
  ('NERNST POTENTIAL AT FUEL CELL EXIT=',E,' V')
  (COL(10),A,E(12,5),A)
  ('FUEL UTILIZATION=',FUEL_UTIL)
  (SKIP(2),COL(10),A,E(12,5))
  ('OXYGEN UTILIZATION=',O2_UTIL)
  (COL(10),A,E(12,5));
END PAFC;

```

A.22 PUMP MODEL

A.22.1 Description of Model

The pump modeled by PUMP handles liquids and requires one pass-through flow of the generic type LIQ. Rather than modeling the exact energy changes through the pump by iterating over the property procedures, PUMP uses an approximation. It calculates the power required as the change in pressure, divided by the density, times the efficiency. The exit enthalpy of the flow is obtained by adding this required power to the inlet flow enthalpy.

The parameters of the PUMP model are as follows:

EXIT_PRES -- Specified exit pressure of the flow.

EFFICIENCY -- Specified efficiency of the pump compression.

A.22.2 Declaration Structure

```
* PROCESS NAME('PUMPC');
PUMPC: PROC( PUMP_P, FLOW_P );

  DCL (PUMP_P, FLOW_P) POINTER;
  DCL 1 PUMP BASED(PUMP_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 EXIT PRES FLOAT(16),
    3 EFFICIENCY FLOAT(16),
    2 POWER,
    3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
    2 COST FLOAT(16);
  DCL FLOW BASED(FLOW_P) LIKE FLC;
  DCL
    (T,S,R,Q) FLOAT(16),
    GP ENTRY;
```

Initialize the power to zero.

```
POWER=0.0;
```

Calculate the power consumed, using the incompressible-fluid approximation and the specified exit pressure and efficiency.

```
POWER.CONSUMED=FLOW.MASS*
  1.01325E5*(EXIT_PRES-FLOW.PRES)/(FLOW.RHO*EFFICIENCY);
```

Adjust the exit enthalpy to reflect the power consumed.

```
FLOW.ENTH=FLOW.ENTH+POWER.CONSUMED/FLOW.MASS;
```

Set the flow's exit pressure and call the properties code to obtain the flow's other state variables.

```
FLOW.PRES=EXIT_PRES;
CALL GP(NAME,FLOW,10B);
COST=0.0;
```

Save the exit flow.

```
FLC = FLOW;
RETURN;

PUMPOUT: ENTRY(PUMP_P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('EXIT PRESSURE = ',EXIT_PRES,
  'EFFICIENCY = ',EFFICIENCY)
  (COL(10),A,E(12,5));
END PUMPC;
```

A.23 PIPE-CALCULATOR MODEL

A.23.1 Description of Model

The pipe-calculator (PIC) model is common to all models.

A.23.2 Declaration Structure

```
* PROCESS NAME('PIC');
PIC: PROC(PI_P, FLOW_P);

DCL (PI_P, FLOW_P) POINTER;
DCL 1 PI_BASED(PI_P),
  2 NAME CHAR(16),
  2 FLC,
  3 FNAME CHAR(16),
  3 ID CHAR(16) VARYING,
  3 ATOM(8) FLOAT(16),
  3 PROP,
```

```

      4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
  4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
    XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
  4 WTF FLOAT(16),
2 PARM,
  3 MODE CHAR(10),
  3 PRES_DROP_FRAC FLOAT(16),
  3 FLOW_FACT FLOAT(16);
DCL FLOW BASED(FLOW_P) LIKE FLC;
DCL (GP) ENTRY;
FLC=FLOW;
IF MODE='DESIGN' THEN
DO;
  FLOW.PRES=FLOW.PRES-PRES_DROP_FRAC*FLOW.PRES;
  FLOW_FACT=(FLC.PRES-FLOW.PRES)*FLC.RHO/FLOW.MASS**2;
END;
ELSE
DO;
  PRES_DROP_FRAC=FLOW_FACT*FLOW.MASS**2/(FLC.RHO*FLOW.PRES);
  FLOW.PRES=FLOW.PRES-PRES_DROP_FRAC*FLOW.PRES;
END;
CALL GP(NAME,FLOW,10B);
FLC=FLOW;
RETURN;

PIOUT: ENTRY(PI P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('MODE = ',MODE,
  'FRACTIONAL PRESSURE DROP = ',PRES_DROP_FRAC,
  'FLOW FACTOR = ',FLOW_FACT)
  (COL(10),A,A,2 (SKIP,COL(10),A,E(13,5)));
END PIC;

```

A.24 STEAM-CONDENSER MODEL

A.24.1 Description of Model

Any of the condensible fluids (in addition to steam) may be used with the steam-condenser model (SC). The model requires only one pass-through flow, representing both the input flow and the condensed output flow. This flow is of the generic type STM. The energy extracted by the condensing process is saved in the POWER substructure.

The only parameter of the SC model is EXIT_PRES, the specified exit pressure of the model. If EXIT_PRES is set to zero, then the exit pressure is assumed to be equal to the inlet pressure.

A.24.2 Declaration Structure

```

PROCESS NAME('SCC');
SCC: PROC( SC_P, STM_P);

    DCL (SC_P, STM_P) POINTER;
    DCL 1 SC BASED(SC_P),
        2 NAME CHAR(16),
        2 FLC,
        3 FNAME CHAR(16),
        3 ID CHAR(16) VARYING,
        3 ATOM(8) FLOAT(16),
        3 PROP,
        4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
        3 COMP,
        4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
          XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
        3 SOL,
        4 WTF FLOAT(16),
        2 PARM,
        3 EXIT_PRES FLOAT(16),
        2 POWER,
        3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
        2 COST FLOAT(16);
    DCL STM BASED(STM_P) LIKE FLC;
    DCL (PC, ENTH_LIQ, ENTH_STM, T_SAT) FLOAT(16),
        (GP,GPSAT) ENTRY;

```

Set the power to zero and save the inlet flow.

```

POWER=0.0;
FLC = STM;

```

If the specified exit pressure is not zero, set the exit pressure of the flow; if this is not done, the exit pressure will be the same as the inlet pressure.

```

IF PARM.EXIT_PRES=0.0 THEN
    STM.PRES=PARM.EXIT_PRES;

```

Call GPSAT to obtain the saturation enthalpy value.

```

CALL GPSAT(NAME,STM,PC,ENTH_LIQ,ENTH_STM);

```

Calculate the power loss necessary to bring the flow down to the saturation liquid enthalpy level.

```

POWER.LOSS = STM.MASS*(STM.ENTH-ENTH_LIQ);

```

Set the exit flow enthalpy equal to the saturation value.

```

STM.ENTH = ENTH_LIQ;

```

Call the properties code to obtain the other exit flow conditions.

```
CALL GP(NAME,STM,10B);
COST=0.0;
```

Save the exit flow.

```
FLC = STM;
RETURN;

SCOUT: ENTRY(SC P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('EXIT PRESSURE = ',EXIT_PRES)
(COL(10),A,E(12,5));
END SCC;
```

A.25 STEAM-DRUM MODEL

A.25.1 Description of Model

The steam-drum model (SD) requires two flows, both of the generic type STM. The first is a pass-through flow, representing the two-phase input flow and, as an output, the downcomer flow. The second flow is an output flow, representing the saturated-steam flow. The model is a demand-type model; the input flow must be of a specified steam quality.

The parameters of the SD model are as follows:

QUAL -- Specified steam quality of the input flow.

CONS -- Difference between the enthalpy of the incoming flow and that required by the specified steam quality. This parameter is calculated by the model to aid in obtaining the correct input steam quality. Thus, CONS should be constrained to equal zero outside the model.

A.25.2 Declaration Structure

```
* PROCESS NAME('SDC');
SDC: PROC( SD_P, STM1_P, STM2_P);

DCL (SD_P, STM1_P, STM2_P) POINTER;
DCL 1 SD BASED(SD_P),
2 NAME CHAR(16),
2 FLC1,
3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
```

```

3  ATOM(8) FLOAT(16),
3  PROP,
4  (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
3  COMP,
4  (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
   XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3  SOL,
4  WTF FLOAT(16),
2  FLC2,
3  FNAME CHAR(16),
3  ID CHAR(16) VARYING,
3  ATOM(8) FLOAT(16),
3  PROP,
4  (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
3  COMP,
4  (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
   XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3  SOL,
4  WTF FLOAT(16),
2  PARM,
3  QUAL FLOAT(16),
3  CONS FLOAT(16),
2  COST FLOAT(16);
DCL STM1 BASED(STM1_P) LIKE FLC1;
DCL STM2 BASED(STM2_P) LIKE FLC2;
DCL (PC, ENTH_ST, ENTH_LIQ) FLOAT(16),
    TNAME CHAR(16),
    (GP,GPSAT) ENTRY;

```

Save the incoming flow.

```
FLC1=STM1;
```

Initialize the second flow to be the same as the first. (The flow name, stored within the variable FNAME, should not be set to that of the first flow; it should be left unchanged, so it is temporarily saved in TNAME.

```

TNAME=STM2.FNAME;
STM2=STM1;
STM2.FNAME=TNAME;

```

Call the procedure GPSAT to obtain the critical pressure, as well as the saturation liquid and vapor enthalpy values.

```
CALL GPSAT(NAME,STM1,PC,ENTH_LIQ,ENTH_ST);
```

If the pressure of the flow is less than critical, then perform the drum calculations.

```

IF STM1.PRES<PC THEN
DO;

```

In order to make sure that both exit flows always have a nonzero mass flow rate, it is assumed that the inlet flow has an enthalpy level equivalent to that of the specified inlet quality. This difference between the actual inlet enthalpy and that required is evaluated and stored in the variable CONS. This variable will need to be constrained to equal zero.

```
CONS=STM1.ENTH-ENTH_LIQ-PARM.QUAL*(ENTH_ST-ENTH_LIQ);
```

On the basis of the specified quality, calculate the mass flow rates of the two exit flows.

```
STM2.MASS=PARM.QUAL*STM1.MASS;
STM1.MASS=STM1.MASS-STM2.MASS;
```

Set the enthalpies of the exit flows to the saturation liquid and vapor values.

```
STM2.ENTH=ENTH_ST;
STM1.ENTH=ENTH_LIQ;
STM2.VEL=STM1.VEL;
STM1.QUAL=0.0;
```

Call the properties code to obtain the saturation temperature of the flows.

```
CALL GP(NAME,STM2,10B);
STM1.TEMP=STM2.TEMP;
```

Save the exit flows. (Because the properties code was not recalled for the liquid exit flow, the density and entropy values are not redefined from their inlet values.)

```
FLC1=STM1;
FLC2=STM2;
END;
ELSE
DO;
  PUT SKIP EDIT(' DRUM BEING CALLED FOR A SUPERCRITICAL FLOW.')
    (COL(2),A);
  STOP;
END;
RETURN;
SDOUT: ENTRY(SD P);
  PUT SKIP EDIT(' ',NAME)(COL(4),A);
  PUT SKIP(2) EDIT('QUALITY = ',PARM.QUAL)
    (COL(10),A,E(12,5));
END SDC;
```

A.26 LIQUID-GAS SEPARATOR MODEL

A.26.1 Description of Model

The two-phase, two-component liquid-gas separator model (SEPR) requires two pass-through flows and two output flows, all of the generic type GAS. The first pass-

through flow represents the predominant gaseous component, while the second represents the liquid component. The first of the output flows represents any gaseous carry-over flow that leaves with the liquid pass-through flow. The second output flow represents any liquid carry-over that leaves the separator with the gaseous pass-through flow.

The parameters of the SEPR model are as follows:

VELOCITY_HEAD_RATIO -- Specified ratio of the square of the liquid velocity out of the separator to the square of the liquid velocity into the separator (used only when the separator is run in the specified efficiency mode).

PRES_DROP(2) -- Specified array of pressure drops for the gaseous and liquid flows through the separator.

VOL_RATIO -- Calculated ratio of the volume flow rate for the gas to that for the liquid.

LIQ_CO -- Specified fraction of the liquid flow carried over with the gas flow.

GAS_CO -- Specified fraction of the gas flow carried over with the liquid flow.

EFFICIENCY -- Calculated ratio of the fraction of liquid mass to total mass, times the velocity head ratio.

VAPOR_CO -- Calculated mass of liquid carried over with the gas flow due to the vaporization of the liquid. This parameter, calculated when VAPOR_INC is set to "YES," is used only to indicate how much of the liquid vapor is carried over. The actual mass is not combined with the exiting gas flow or subtracted from the liquid flow.

VAPOR_INC -- Switch used to indicate whether or not liquid-vapor carry-over is to be calculated.

HEAT_REJECTED -- Calculated heat that would be lost from the liquid if vapor carry-over occurred.

A.26.2 Declaration Structure

```
* PROCESS NAME('SEPR');
SEPRC: PROC(SEPR_P,GAS_P,LIQ_P,GCO_P,LCO_P);

DCL (SEPR_P,GAS_P,LIQ_P,GCO_P,LCO_P) POINTER;
DCL 1 SEPR BASED(SEPR_P),
    2 NAME CHAR(16),
    2 FLCL,
```

```

3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 FLC2,
3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 FLC3,
3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 FLC4,
3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 PARM,
3 VELOCITY HEAD RATIO FLOAT(16),
3 PRES DROP(2) FLOAT(16),
3 VOL RATIO FLOAT(16),
3 LIQ_CO FLOAT(16),
3 EFFICIENCY FLOAT(16),
3 GAS_CO FLOAT(16),
3 VAPOR INC CHAR(3),
3 VAPOR_CO FLOAT(16),
3 HEAT REJECTED FLOAT(16),
2 POWER,

```

```

      3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL GAS BASED(GAS_P) LIKE FLC1;
DCL LIQ BASED(LIQ_P) LIKE FLC2;
DCL GCO BASED(GCO_P) LIKE FLC3;
DCL LCO BASED(LCO_P) LIKE FLC4;
DCL (GP,VPSOLN,HFGSOLN,VISSOLN,OPT1) ENTRY;
DCL TNAME CHAR(16);
DCL (MW_VAPOR,MW_GAS,P_VAPOR,HFG_VAPOR) FLOAT(16);
DCL (SQRT) BUILTIN;

```

Assume GAS.QUAL = 1.0 and LIQ.QUAL = 0.0: Save the inlet flows.

```

FLC1=GAS;
FLC2=LIQ;

```

Initialize the power to zero.

```

POWER=0.0;

```

Redefine the exit pressures based on any specified pressure drops.

```

GAS.PRES=FLC1.PRES - PARM.PRES_DROP(1);
LIQ.PRES=FLC2.PRES - PARM.PRES_DROP(2);

```

Set the exit velocity of the liquid flow, based on the specified velocity-head ratio.

```

LIQ.VEL=FLC2.VEL*SQRT(PARM.VELOCITY_HEAD_RATIO);

```

Set the exit velocity of the gas equal to that of the liquid.

```

GAS.VEL=LIQ.VEL;

```

Maintaining conservation of energy, calculate the exit enthalpies of the gas and liquid flows.

```

GAS.ENTH=FLC1.ENTH + 0.5*(FLC1.VEL**2 - GAS.VEL**2);
LIQ.ENTH=FLC2.ENTH + 0.5*(FLC2.VEL**2 - LIQ.VEL**2);

```

Call the properties code for both flows to determine the other state variables of the flows.

```

CALL GP(NAME,GAS,10B);
CALL GP(NAME,LIQ,10B);

```

Initialize the state conditions of the carry-over flows. (The flow's name should not be disturbed in this initialization, so it is temporarily saved in TNAME.)

```

TNAME=LCO.FNAME;
LCO=LIQ;
LCO.FNAME=TNAME;
TNAME=GCO.FNAME;

```

```
GCO=GAS;
GCO.FNAME=TNAME;
```

On the basis of the specified gas and liquid carry-over fractions, set the carry-over mass flow rates.

```
LCO.MASS=LIQ_CO*FLC2.MASS;
LIQ.MASS=LIQ.MASS - LCO.MASS;
GCO.MASS=GAS_CO*FLC1.MASS;
GAS.MASS=GAS.MASS - GCO.MASS;
IF VAPOR_INC = 'YES' THEN
  DO;
```

If a calculation of liquid-vapor carry-over is desired, first set the value of the gas's molecular weight, assuming an ideal gas law.

```
MW_GAS=GAS.RHO*GAS.TEMP*8314.3/(GAS.PRES*101325.0);
```

Call the vapor pressure routine for liquids to determine the liquid vapor pressure at the liquid's temperature and to obtain the liquid's molecular weight.

```
CALL VPSOLN(LIQ.ID,LIQ.TEMP,P_VAPOR,MW_VAPOR);
```

Calculate the mass flow rate of any liquid-vapor carry-over.

```
VAPOR_CO=GAS.MASS*MW_VAPOR*P_VAPOR/(MW_GAS*(GAS.PRES-P_VAPOR));
```

Call the latent heat of vaporization procedure to determine the heat of vaporization of the liquid. Determine the total heat rejected due to the vapor carry-over.

```
CALL HFGSOLN(LIQ.ID,LIQ.TEMP,HFG_VAPOR);
HEAT_REJECTED=VAPOR_CO*HFG_VAPOR;
```

Readjust the liquid enthalpy due to the vapor carry-over loss. Call the properties code to obtain the new state conditions of the flow. (The vapor carry-over calculations are only approximate, since the code cannot at present mix this vapor with the gas-side flow. The liquid mass also is not readjusted for carry-over loss.)

```
LIQ.ENTH=LIQ.ENTH-HEAT_REJECTED/LIQ.MASS;
CALL GP(NAME,LIQ,10B);
END;
ELSE
  DO;
```

If vapor carry-over calculations are not desired, set variables for printout.

```
VAPOR_CO=0.0;
HEAT_REJECTED=0.0;
END;
```

Define any power loss due to vapor carry-over. (For most liquid-metal systems, the vapor carry-over would appear on the gas side; thus, this power loss may not be actual.)

```
POWER.LOSS=HEAT_REJECTED;
```

Calculate the efficiency of the separation process, based on the mass flow rates and velocities.

```
EFFICIENCY=LIQ.MASS/(LIQ.MASS + GAS.MASS)*  
(LIQ.VEL/FLC2.VEL)**2;
```

Save the exit flows.

```
FLC1=GAS;  
FLC2=LIQ;  
FLC3=GCO;  
FLC4=LCO;  
RETURN;  
SEPROUT: ENTRY(SEPR_P);  
PUT SKIP EDIT(' ',NAME)(COL(4),A);  
PUT EDIT('VEL HEAD RATIO=',VELOCITY_HEAD_RATIO,  
'PRES DROP(1)=' ,PRES_DROP(1),  
'PRES DROP(2)=' ,PRES_DROP(2),  
'EFFICIENCY=' ,EFFICIENCY,  
'VAPOR C/O=' ,VAPOR_CO,  
'HEAT REJECTED=' ,HEAT_REJECTED,  
'LIQUID C/O=' ,LIQ_CO,  
'GAS C/O=' ,GAS_CO)  
(COL(10),A,E(12,5));  
END SEPRC;
```

A.27 STACK MODEL

A.27.1 Description of Model

The stack model (SK) requires one pass-through flow of the generic type GAS. The parameters of the SK model are as follows:

A_TEMP -- Specified ambient temperature at the stack exit.

A_PRES -- Specified ambient pressure at the stack exit.

A.27.2 Declaration Structure

```

* PROCESS NAME('SKC');
SKC: PROC( SK_P, GAS_P);

  DCL (SK_P, GAS_P) POINTER;
  DCL 1 SK BASED(SK_P),
    2 NAME CHAR(16),
    2 FLC,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 A_TEMP FLOAT(16),
    3 A_PRES FLOAT(16),
    3 ENERGY_REJECTED FLOAT(16),
    2 POWER,
    3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
    2 COST FLOAT(16);
  DCL GAS BASED(GAS_P) LIKE FLC,
    (GP) ENTRY;

```

Save the inlet flow to the stack.

```
FLC = GAS;
```

Set the exit conditions of the gas from the stack to match the ambient conditions.

```

GAS.TEMP=A_TEMP;
GAS.PRES=A_PRES;

```

Call the properties code to determine the other state conditions at the stack exit.

```
CALL GP(NAME,GAS,1B);
```

Calculate the energy loss from the stack.

```
LOSS,ENERGY_REJECTED = FLC.MASS*(FLC.ENTH+0.5*FLC.VEL**2-GAS.ENTH);
```

Save the exit flow from the stack.

```

FLC=GAS;
COST=0.0;
RETURN;

```

```

SKOUT: ENTRY(SK_P);
  PUT SKIP EDIT(' ',NAME)(COL(4),A)
    ('ENERGY REJECTED = ',ENERGY_REJECTED,' W')
    (COL(10),A,E(12,5),A);
END SKC;

```

A.28 SOLID-OXIDE FUEL-CELL MODEL

A.28.1 Description of Model

The solid-oxide fuel-cell model (SOFC) requires two flows, both of the generic type GAS. The first flow represents the anode flow; the second, the cathode flow.

The parameters of the SOFC model are as follows:

CELL_CURRENT -- Specified current through each cell.

CELL_VOLTAGE -- Calculated cell voltage.

CELL_TEMP -- Specified average temperature of a cell.

STACK_VOLTAGE -- Calculated total voltage across the cell stack.

NO_OF_CELLS -- Specified total number of cells in the stack.

DELTA_VOLT -- Specified difference between the Nernst potential at the fuel-cell exit and the cell voltage.

FUEL_UTIL -- Calculated value of the fuel utilization.

O2_UTIL -- Calculated value of the O₂ utilization.

HF -- Calculated value of the overall isothermal heat of reaction.

E -- Calculated Nernst potential at the fuel-cell exit.

A.28.2 Declaration Structure

```

* PROCESS NAME('SOFCC');
SOFCC: PROC(SOFC_P,FGAS1_P,FGAS2_P);

  DCL (SOFC_P,FGAS1_P,FGAS2_P) POINTER;
  DCL 1 SOFC BASED(SOFC_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),

```

```

3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,NS,NCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 FLC2,
3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,NS,NCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 PARM,
3 CELL_CURRENT FLOAT(16),
3 CELL_VOLTAGE FLOAT(16),
3 (STACK_VOLTAGE,CELL_TEMP) FLOAT(16),
3 NO_OF_CELLS FLOAT(16),
3 DELTA_VOLT FLOAT(16),
3 FUEL_UTIL FLOAT(16),
3 O2_UTIL FLOAT(16),
3 (HF,E) FLOAT(16),
2 POWER,
3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL FGAS1 BASED(FGAS1_P) LIKE FLC1, FGAS2 BASED(FGAS2_P) LIKE FLC2;
DCL (Y,O1 MOL,O2 MOL,H2 MOL,H DEL TC,MOL WT,
H DEL,TTRY,HAIN,HAIN_TC,HAOUT,HAOUT_TC,F INIT(96487.E0),
E0,P H2O AN,P CO2 AN,P H2 AN,P CO2 CA,P O2 CA,
HCIN,HCIN_TC,HCOUT,HCOUT_TC,GRÖSS_POWER,CH4_REQ) FLOAT(16);
DCL (GASBW,GASMR,GP) ENTRY;
DCL SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*));

```

Perform the anode calculations: Initialize the power to zero.

POWER = 0.0;

Save the value of the inlet anode flow enthalpy.

HAIN = FGAS1.ENTH*FGAS1.MASS;

Set the anode flow temperature to the specified cell temperature. Call the properties code to obtain the state conditions of the flow at this temperature.

```

FGAS1.TEMP = CELL_TEMP;
CALL GP(NAME,FGAS1,1B);

```

Save the value of the anode flow enthalpy at this cell temperature.

```
HA1N_TC = FGAS1.ENTH*FGAS1.MASS;
```

The fuel cell only works with the gas-flow streams, so any entrained-solid flow is temporarily subtracted and saved in the FLC1 flow variables.

```
FLC1.WTF=FGAS1.WTF*FGAS1.MASS;  
FGAS1.MASS=FGAS1.MASS-FLC1.WTF;
```

The procedure GASMR calculates the molar flow rates of the flow, given the mass flow rate and the species' molar fractions, if its first argument is 1. These molar flow rates are temporarily stored in the FLC1 variables.

```
CALL GASMR(1B,FGAS1.COMP,FGAS1.MASS,FLC1.COMP,MOL_WT);
```

Given the cell current and number of cells, calculate the total number of moles of O₂ that will cross over from the cathode stream. This species is then added to that already within the anode stream.

```
O1_MOL = CELL_CURRENT*NO_OF_CELLS/(2000.0*F);  
O2_MOL = O1_MOL/2.0;  
FLC1.XO2 = FLC1.XO2+O2_MOL;
```

Calculate the hydrogen concentration within the anode flow.

```
H2_MOL = 4.0*FLC1.XCH4+FLC1.XH2+FLC1.XCO;
```

Check whether or not there is sufficient hydrogen concentration within the anode flow. If not, indicate how much more is needed and terminate.

```
IF (H2_MOL < O1_MOL) THEN  
DO;  
  CH4_REQ = O1_MOL*1.25/4.0 - FLC1.XH2;  
  PUT SKIP(2) EDIT('** ERROR IN SOFC:',  
    'NOT ENOUGH FUEL IN ANODE INLET STREAM.',  
    'INCREASE CH4 FLOW RATE TO ',CH4_REQ,  
    '(MOL/S) OR MORE.')  (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,  
    E(13,5),X(2),A);  
STOP;  
END;
```

Calculate the fuel utilization.

```
FUEL_UTIL = O1_MOL/H2_MOL;
```

Recalling the GASMR procedure (with its first argument set to 2), convert the molar flow rates back to species' molar fractions and obtain the total mass flow rate. These values are placed in the FGAS1 flow variables.

```
CALL GASMR(10B,FGAS1.COMP,FGAS1.MASS,FLC1.COMP,MOL_WT);
```

Given the new values of the FGAS1 composition, call GASBW to obtain the ATOM array.

```
CALL GASBW(FGAS1.COMP,FGAS1.ATOM);
```

Add any entrained solids back to the flow. Call the properties code to obtain the exit flow conditions.

```
FGAS1.MASS=FGAS1.MASS+FLC1.WTF;
FGAS1.WTF=FLC1.WTF/FGAS1.MASS;
CALL GP(NAME,FGAS1,1B);
```

Save the exit flow enthalpy from the anode.

```
HAOUT_TC = FGAS1.ENTH*FGAS1.MASS;
```

Next, perform the cathode calculations: Save the cathode inlet enthalpy.

```
HCIN = FGAS2.ENTH*FGAS2.MASS;
```

Set the cathode flow temperature to equal the cell temperature.

```
FGAS2.TEMP = CELL TEMP;
CALL GP(NAME,FGAS2,1B);
HCIN_TC = FGAS2.ENTH*FGAS2.MASS;
```

Subtract any entrained solids and call GASMR to obtain the molar flow rates, as was done for the anode stream.

```
FLC2.WTF=FGAS2.WTF*FGAS2.MASS;
FGAS2.MASS=FGAS2.MASS-FLC2.WTF;
CALL GASMR(1B,FGAS2.COMP,FGAS2.MASS,FLC2.COMP,MOL_WT);
```

Adjust the species' molar rates to reflect the crossover O_2 . Calculate the O_2 utilization rate.

```
O2_UTIL = O2_MOL/FLC2.XO2;
FLC2.XO2 = FLC2.XO2-O2_UTIL;
```

Check whether or not there is sufficient O_2 with the stream. If not, indicate how much would be needed and terminate.

```
IF (FLC2.XO2 < 0.0) THEN
  DO;
    PUT SKIP(2) EDIT('** ERROR IN SOFC:',
      'NOT ENOUGH O2 IN CATHODE INLET STREAM.',
      'INCREASE O2 MOLE RATE TO ',1.25*O2_MOL,
      '(MOL/S) OR MORE')
      (SKIP(2),2 (COL(4),A,SKIP(1)),COL(4),A,E(13,5),
      X(2),A);
    STOP;
  END;
```

Convert back to molar fractions, set the new ATOM array values, add the subtracted entrained solids, and call the properties code.

```
CALL GASMR(10B,FGAS2.COMP,FGAS2.MASS,FLC2.COMP,MOL_WT);
CALL GASBW(FGAS2.COMP,FGAS2.ATOM);
FGAS2.MASS=FGAS2.MASS+FLC2.WTF;
FGAS2.WTF=FLC2.WTF/FGAS2.MASS;
CALL GP(NAME,FGAS2,1B);
```

Save the exit cathode flow enthalpy.

```
HCOUT_TC = FGAS2.ENTH*FGAS2.MASS;
```

Finally, perform the energy balance calculations: Save the total enthalpy change across the cell (at the cell temperature).

```
H_DEL_TC=HAOUT_TC-HAIN_TC+HCOUT_TC-HCIN_TC;
HF=-H_DEL_TC;
```

The actual exit temperature of the cell will be determined by varying the exit temperature until the total enthalpy change across the cell stack represents the gross power produced by the cell. For the first try, the exit temperature is taken as equal to the cell temperature. Thereafter, it is controlled by the one-dimensional equation solver, SOV.

```
TTRY=CELL_TEMP;
DO I=1 TO 25;
  TTRY=MAX(MIN(2.0*CELL_TEMP,TTRY),0.6*CELL_TEMP);
```

Set the anode exit temperature and call the properties code to obtain the exit enthalpy.

```
FGAS1.TEMP = TTRY;
CALL GP(NAME,FGAS1,1B);
HAOUT = FGAS1.ENTH*FGAS1.MASS;
```

Set the cathode exit temperature and call the properties code to obtain the exit enthalpy.

```
FGAS2.TEMP = TTRY;
CALL GP(NAME,FGAS2,1B);
HCOUT = FGAS2.ENTH*FGAS2.MASS;
```

Calculate the total enthalpy change across the cell.

```
H_DEL=HAOUT-HAIN+HCOUT-HCIN;
STACK_VOLTAGE=NO_OF_CELLS*(0.920-DELTA_VOLT+4.3047E-5*CELL_TEMP*
(0.5*LOG(FGAS1_PRES)+LOG(FGAS1.XH2*FGAS2.XO2**0.5/FGAS1.XH2O)));
GROSS_POWER = STACK_VOLTAGE*CELL_CURRENT;
```

The difference in enthalpy change and gross power is saved in the variable Y. The equation solver is called to obtain a new estimate of TTRY, the exit temperature.

```

Y=-H_DEL-GROSS_POWER;
CALL SOV(TTRY,Y,I,100.0,0.1,25,0,'SOFC_T');
END;
CELL_VOLTAGE = STACK_VOLTAGE/NO_OF_CELLS;

```

The power produced is saved for printout, and the exit flow conditions are saved.

```
POWER.PRODUCED = GROSS_POWER;
```

Calculate the Nernst potential at the cell exit.

```

P_H2O_AN = FGAS1.XH2O*FGAS1.PRES;
P_H2_AN = FGAS1.XH2*FGAS1.PRES;
P_O2_CA = FGAS2.XO2*FGAS2.PRES;
E0 = .021682E0*(57.939E0-FGAS1.TEMP*(11.527E-3+.6E-6*FGAS1.TEMP));
E = E0+43.086E-6*(LOG(P_H2_AN/(P_H2O_AN))
    *FGAS1.TEMP+.5E0*LOG(P_O2_CA)*FGAS2.TEMP);
FLC1=FGAS1;
FLC2=FGAS2;
RETURN;
SOFCOUT: ENTRY(SOFC_P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT
('CELL TEMPERATURE = ',CELL_TEMP,' K')
(SKIP(2),COL(10),A,E(12,5),A)
('CELL CURRENT = ',CELL_CURRENT,
 ' A') (COL(10),A,E(12,5),A)
('CELL VOLTAGE = ',CELL_VOLTAGE,
 ' V') (COL(10),A,F(17,5),A)
('NO OF CELLS = ',NO_OF_CELLS)
(COL(10),A,F(17,0))
('STACK VOLTAGE = ',STACK_VOLTAGE,' V')
(COL(10),A,E(12,5),A)
('OVERALL ISOTHERMAL HEAT OF REACTION = ',HF,
 ' W') (COL(10),A,E(12,5),A)
('STACK GROSS POWER = ',POWER.PRODUCED,' W')
(COL(10),A,E(11,5),A)
('NERNST POTENTIAL AT FUEL CELL EXIT = ',E,' V')
(COL(10),A,E(12,5),A)
('FUEL UTILIZATION = ',FUEL_UTIL)
(SKIP(2),COL(10),A,E(12,5))
('OXYGEN UTILIZATION = ',O2_UTIL)
(COL(10),A,E(12,5));
END SOFC;

```

A.29 FLOW-SPLITTER MODEL

A.29.1 Description of Model

The flow-splitter model (SP) requires one pass-through flow, representing the input flow and (on output) one of the two output flows. A second flow to the model

represents the second output flow. Both of these flows are of the generic type GAS. The SP model provides options for splitting the flow not only by mass, but also by composition. Thus, the SP model can be used to model processes that split off specific species of the input flow.

The parameters of the SP model are as follows:

SPLIT_RATIO -- Specified fraction of the input flow mass split off into the second flow. If **SPLIT_RATIO** is set, then **SR** (see below) should not be used.

SPLIT_MASS -- Specified portion of mass flow rate split off into the second flow, if set to be greater than zero. If this variable is set to zero, then the mass flow rate split off is defined by **SPLIT_RATIO**. **SPLIT_MASS** must be less than the mass flow rate of the input flow.

SR -- Specified structure variable representing the split ratios by weight fractions of the species flow rates split off into the second flow. The elements of this structure are the same as those of the **COMP** substructure within the generic GAS flow, but without the "X" prefix (e.g., AR, CH₄, O₂, SO₂). If the **SR** structure is specified, then the mass flow rate of the second flow is determined by the sum of the flow rates of the individual species. **SR** should not be specified if **SPLIT_RATIO** is used.

A.29.2 Declaration Structure

```
* PROCESS NAME('SPC');
SPC: PROC( SP_P, F1_P, F2_P);

DCL (SP_P, F1_P, F2_P) POINTER;
DCL 1 SP BASED(SP_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
```

```

      4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
      4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
      4 WTF FLOAT(16),
2 PARM,
      3 SPLIT_RATIO FLOAT(16),
      3 SPLIT_MASS FLOAT(16),
      3 SR,
      4 (AR,CH4,CO,CO2,H,H2,H2O,H2S,K,KOH,NO,N2,
      O,OH,O2,SO2,HCL,CH3OH,C,COS,NH3,S,CL) FLOAT(16),
      3 SR SOL FLOAT(16),
      3 POWER_REQUIRED FLOAT(16),
      2 COST FLOAT(16);
DCL (GASBW,GASNM,GP,GPSAT) ENTRY;
DCL SPECMW LIKE FLC1.COMP BASED(MW_PTR);
DCL MW(23) FLOAT(16) INIT(39.948,16.04303,28.01055,44.00995,1.00797,
      2.01594,18.01534,34.07994,39.102,56.10937,30.0061,28.0134,
      15.9994,17.00737,31.9988,64.0628,36.460974,32.04243,12.01115,
      60.07455,17.03151,32.064,35.453);
DCL (A_PTR,MW_PTR) POINTER;
DCL ARRY(23) FLOAT(16) BASED(A_PTR);
DCL (NM,PCRIT,HL,HV) FLOAT(16);
DCL TNAME CHAR(16);
DCL F1 BASED(F1_P) LIKE FLC1,
      F2 BASED(F2_P) LIKE FLC2;
MW_PTR=ADDR(MW);

```

Initialize power required to zero.

```
POWER_REQUIRED=0.0;
```

Initialize second flow to be the same as the first flow. (The second flow's FNAME should not be changed during this initialization; it is temporarily saved in TNAME.)

```

FLC1 = F1;
TNAME=F2.FNAME;
F2=F1;
F2.FNAME=TNAME;
IF SPLIT_RATIO>0.0 | SPLIT_MASS>0.0 THEN
  DO;

```

If the flow is being split via a split ratio rather than by species ratios, then only the mass flow rates need to be readjusted. If the SPLIT_MASS option is used, then the split ratio is determined first.

```

  IF SPLIT_MASS>0.0 THEN
    SPLIT_RATIO=SPLIT_MASS/F1.MASS;
    F2.MASS=SPLIT_RATIO*F1.MASS;
    F1.MASS=F1.MASS-F2.MASS;
  END;

```

```
ELSE
DO;
```

If the flow is being split via species split ratios, then the mass flow rates of the individual species must be calculated first. A call is made to GASNM to determine the total number of moles of the gas flow.

```
CALL GASNM(F1.COMP,NM);
```

The mass flow rates of the split-off species are determined based on the individual split ratios, the species' molecular weights, and the flow's mass flow rate, minus any entrained-solids flow.

```
F2.COMP=SR*NM*F1.COMP*SPECMW*F1.MASS*(1.-F1.WTF);
```

The mass flow rates of the remaining flow are then determined in similar fashion.

```
F1.COMP=(1.0-SR)*NM*F1.COMP*SPECMW*F1.MASS*(1.0-F1.WTF);
```

The split between any entrained solids is then calculated, based on the specified split ratio (SR) for the solids.

```
F2.WTF=SR_SOL*F1.WTF*F1.MASS;
F1.WTF=(1.0-SR_SOL)*F1.WTF*F1.MASS;
```

With the calculated mass flow rates of the individual species for the second flow, the total mass flow rate is determined by summation. The molar flow rates are then determined by dividing the mass flow rates by the species' molecular weights. The total number of moles is obtained by summing the molar flow rates. Finally, the molar fractions are obtained by dividing the molar rates by the total number of moles.

```
A_PTR=ADDR(F2.COMP);
F2.MASS=SUM(ARRY);
F2.COMP=F2.COMP/SPECMW;
NM=SUM(ARRY);
IF NM>0.0 THEN
  F2.COMP=F2.COMP/NM;
ELSE
  F2.COMP=0.0;
```

The "save" sequence of calculations performed on the second flow are now performed on the first flow to obtain the new molar fractions for this flow.

```
A_PTR=ADDR(F1.COMP);
F1.MASS=SUM(ARRY);
F1.COMP=F1.COMP/SPECMW;
A_PTR=ADDR(F1.COMP);
NM=SUM(ARRY);
IF NM>0.0 THEN
  F1.COMP=F1.COMP/NM;
ELSE
  F1.COMP=0.0;
```

The procedure GASBW is called for each flow to obtain the new ATOM array for each.

```
CALL GASBW(F1.COMP,F1.ATOM);
CALL GASBW(F2.COMP,F2.ATOM);
```

The solid weight fractions and mass flow rates (including the solids) are calculated. The properties code is called for each flow.

```
F1.MASS=F1.MASS+F1.WTF;
F1.WTF=F1.WTF/F1.MASS;
F2.MASS=F2.MASS+F2.WTF;
F2.WTF=F2.WTF/F2.MASS;
CALL GP(NAME,F1,1B);
CALL GP(NAME,F2,1B);
```

The required power is calculated, representing the enthalpy difference between the entering and exiting flows.

```
POWER REQUIRED=F1.ENTH*F1.MASS+F2.ENTH*F2.MASS
              -FLC1.MASS*FLC1.ENTH;
END;
IF ABS(F2.XH2O-1.0)<1E-4 & F1.ID='GAS' THEN
DO;
```

If the flow split off from a GAS flow happens to be pure water, then the second flow type is set to H2O and is assumed to be on the saturation line. (As was the case with the MX model, this option should be used with caution.)

```
F2.ID='H2O';
CALL GPSAT(NAME,F2,PCRIT,HL,HV);
F2.ENTH=HL;
CALL GP(NAME,F2,10B);
END;
COST=0.0;
```

Save the exit flows.

```
FLC1= F1;
FLC2= F2;
RETURN;
```

```
SPOUT: ENTRY(SP_P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('SPLIT RATIO =',SPLIT_RATIO,
'POWER REQUIRED =',POWER_REQUIRED)
(COL(10),A,E(12,5));
END SPC;
```

A.30 STEAM-TURBINE MODEL

A.30.1 Description of Model

The steam-turbine model (ST) provides options for modeling a typical extraction stage, as well as inlet and exhaust stages. The model requires one pass-through flow and one output flow, representing any extracted flow from the turbine stage. This extracted flow is at the same thermodynamic conditions as the pass-through flow. The extracted flow is extracted at the exit of the turbine stage; turbine trains with multiple extraction points would be modeled using multiple ST models. Both of the flows to the ST model are of the generic type STM.

The model provides for an off-design mode by calculating a flow factor in the design mode that is then used in the off-design mode. This flow factor represents a nondimensional flow rate that the turbine stage can pass. In the off-design mode, the parameter CONS (based on this flow factor) should be constrained to equal zero.

Tables of exhaust-loss enthalpy corrections for use in the exhaust stage may be read in by calling STIN. These tables define the value of the exhaust-loss enthalpy vs. the turbine inlet mass, normalized by dividing by a design-point mass. This exhaust-loss enthalpy is then added to the exit-flow enthalpy from the turbine. (In design-mode calculations, this design-point mass flow rate is assigned the value of the inlet mass.)

For the inlet turbine stage in the design mode, the exit pressure is calculated to give a required steam flow velocity. This required velocity is obtained from a specified turbine-wheel speed and a specified wheel-to-flow-velocity ratio. The efficiency is also calculated, for both the design and off-design modes, as a function of this wheel-to-flow-velocity ratio.

The parameters of the ST model are as follows:

DDNAME -- Specified character string representing the file name that STIN will read to obtain the table of exhaust-loss enthalpies.

MODE -- Specified character string, either "OFF-DESIGN" or "DESIGN." In modeling throttle stages, the characters "IN" may also be appended to the end of the mode string.

EXIT_PRES -- Specified exit pressure of the turbine.

EFFICIENCY -- Specified efficiency of the turbine expansion.

MECH_EFF -- Mechanical efficiency of the turbine. Any thermodynamic energy extracted from the turbine is multiplied by this efficiency to obtain the usable power.

SR -- Specified split ratio of the extracted flow. The mass flow rate of the extracted flow is equal to SR times the input mass flow rate.

EXT_MASS -- Specified extracted mass flow rate. If EXT_MASS is specified, it overrides SR. (EXT_MASS should be less than the inlet mass flow rate.)

FLOW_FACT -- Flow factor, calculated in the design mode and specified in the off-design mode.

EXHAUST_LOSS -- Specified or calculated value of the exhaust-loss enthalpy. If STIN has been called, this value is obtained from the tables; otherwise, the value may be specified as an input.

DM -- Specified value of the design-point mass flow rate in the off-design mode. In the design mode, this parameter is set equal to the inlet mass flow rate to the turbine.

WV -- Specified turbine wheel-to-flow-velocity ratio, used only in the turbine-inlet stage.

WHEEL_SPEED -- Specified turbine-wheel speed, used only in the turbine-inlet stage.

CONS -- Calculated parameter in the off-design mode, representing a measure of the mass flow rate that the turbine stage can pass for the various inlet conditions. This parameter should be constrained to equal zero during off-design calculations.

VOL_FLOW_RATE -- Calculated volume flow rate through the turbine.

PRINT -- Specified switch used to print out iterations within the turbine-inlet-stage option during the calculation of exit pressure.

A.30.2 Declaration Structure

```
* PROCESS NAME('STC');
STC: PROC( ST_P, STM_P, STME_P );

DCL (ST_P, STM_P, STME_P) POINTER;
DCL 1 ST BASED(ST_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP, PRES, ENTH, ENTP, QUAL, RHO, VEL, MASS)    FLOAT(16),
    3 COMP,
    4 (XAR, XCH4, XCO, XCO2, XH, XH2, XH2O, XH2S, XK, XKOH, XNO, XN2,
      XO, XOH, XO2, XS02, XHCL, XCH3OH, XC, XCOS, XNH3, XS, XCL) FLOAT(16),
```

```

      3 SOL,
        4 WTF FLOAT(16),
2 FLC2,
  3 FNAME CHAR(16),
  3 ID CHAR(16) VARYING,
  3 ATOM(8) FLOAT(16),
  3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS)    FLOAT(16),
  3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
  3 SOL,
    4 WTF FLOAT(16),
2 PARM,
  3 DDNAME CHAR(7),
  3 MODE CHAR(15),
  3 EXIT PRES FLOAT(16),
  3 EFFICIENCY FLOAT(16),
  3 MECH EFF FLOAT(16),
  3 SR FLOAT(16),
  3 EXT MASS FLOAT(16),
  3 FLOW_FACT FLOAT(16),
  3 EXHAUST LOSS FLOAT(16),
  3 DM FLOAT(16),
  3 WV FLOAT(16),
  3 WHEEL SPEED FLOAT(16),
  3 CONS FLOAT(16),
  3 VOL FLOW RATE FLOAT(16),
  3 PRINT FIXED BIN(15),
2 POWER,
  3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16),
2 E LOSS,
  3 PTR POINTER,
2 COST FLOAT(16);
DCL STM BASED(STM_P) LIKE FLC1,
  STME BASED(STME_P) LIKE FLC1;
DCL (COV INIT(101325.0), VELC) FLOAT(16),
  (I) FIXED BIN(15),
  TNAME CHAR(16),
  SOV ENTRY(FLOAT(16),FLOAT(16),FIXED BIN(15),FLOAT(16),
    FLOAT(16),FIXED BIN(15),FIXED BIN(15),CHAR(*)),
  NULL BUILTIN,
  (GP, TABC, TABIN) ENTRY;

```

Initialize the power to zero. Call the properties code to make sure the inlet conditions are consistent. (This call should not be necessary, but it has been retained for safety, in case new models are added for which the exiting flows have not been made consistent.

```

POWER = 0.0E0;
CALL GP(NAME,STM,10B);

```

Save the inlet flow.

```

FLC1 = STM;

```

A calculation of the isentropic expansion through the turbine is made first. The exit pressure is set, and a call is made to the properties code with the third argument set to 3 (indicating that the flow's pressure and entropy will be used as the two input state variables).

```
STM.PRES=EXIT_PRES;
CALL GP(NAME,STM,11B);
IF INDEX(MODE,'OFF-DESIGN')=0 THEN
  DO;
```

If the model is being run in the design mode (i.e., the mode is not equal to OFF-DESIGN), set the value of the mass flow rate in DM first and calculate the flow factor to be used in any off-design run.

```
DM=STM.MASS;
FLOW_FACT=STM.MASS*SQRT(MAX(FLC1.TEMP,0.))/(FLC1.PRES*COV);
IF INDEX(MODE,'IN')=0 THEN
  DO;
```

If the mode setting contains the characters IN, then the efficiency of the turbine stage is set based on the specified turbine wheel-to-flow-velocity ratio. In this case, the exit pressure is calculated to represent a throttle process at constant enthalpy. This is done by varying the exit pressure and calling the properties code until the calculated exit enthalpy is equal to the inlet enthalpy. The one-dimensional equation solver, SOV, is used to vary the exit pressure.

```
EFFICIENCY=5.84*WV*(1.0-2*WV);
VELC=WHEEL_SPEED/WV;
STM.ENTH=FLC1.ENTH-0.5*VELC*VELC;
DO I=1 TO 15;
  STM.PRES=EXIT_PRES;
  CALL GP(NAME,STM,10B);
  CALL SOV(EXIT_PRES,STM.ENTP-FLC1.ENTP,I,1.0,1E-6,15,PRINT,
    'ST P');
END;
END;
END;
ELSE
  DO;
```

If the mode is set to OFF-DESIGN, then the flow factor (as calculated in a design-mode run) is used along with the current inlet flow conditions to evaluate CONS. This variable should be constrained to equal zero by suitably varying some variable within the system. (Basically, this constraint represents the choking-flow conditions that will occur within the turbine.) Depending on the system, there may be more than one way in which to force CONS to equal zero. Varying the exit pressure or split ratio from the previous turbine stage are the most common means of constraining CONS to be zero.

```
CONS=FLC1.PRES-FLC1.MASS*SQRT(MAX(FLC1.TEMP,0.))/(FLOW_FACT*COV);
IF INDEX(MODE,'IN')=0 THEN
  DO;
```

If this stage is an inlet stage, the efficiency is calculated based on the ratio of wheel speed to fluid velocity, as in the design mode. For the off-design mode, the exit pressure is specified.

```

      VELC=SQRT(2.*MAX(0.0,FLC1.ENTH-STM.ENTH));
      IF VELC=0.0 THEN
        WV=0.30;
      ELSE
        WV=WHEEL_SPEED/VELC;
        EFFICIENCY=5.84*WV*(1.0-2*WV);
      END;
    END;
  END;

```

If the optional call to STIN has been made, then the exhaust-loss table is used along with the normalized (with respect to the design mass flow rate) mass flow rate to obtain the exhaust-loss enthalpy, which is added to the exit enthalpy for the stage.

```

    IF E_LOSS.PTR=NULL THEN
      CALL TABC(E_LOSS,STM.MASS/DM,EXHAUST_LOSS);

```

The exit enthalpy is calculated, based on the efficiency of the stage and any exhaust-loss enthalpy.

```

      STM.ENTH=EFFICIENCY*(STM.ENTH-FLC1.ENTH)+FLC1.ENTH+EXHAUST_LOSS;

```

A call is now made to the properties code to obtain the other state variables at the exit conditions of pressure and enthalpy.

```

      CALL GP(NAME,STM,10B);

```

For printout, the volume flow rate is calculated. The usable output power is obtained by multiplying the enthalpy drop through the turbine by the mechanical efficiency.

```

      VOL_FLOW_RATE=STM.MASS/STM.RHO;
      POWER.PRODUCED=MECH_EFF*(FLC1.ENTH-STM.ENTH)*FLC1.MASS;

```

Any extraction flow is now calculated, based on either EXT_MASS or SR. The extraction flow is initialized to the same state conditions as the main exit flow.

```

    IF EXT_MASS>0.0 THEN
      SR=EXT_MASS/STM.MASS;
      TNAME=STME.FNAME;
      STME=STM;
      STME.FNAME=TNAME;
      STME.MASS=SR*STM.MASS;
      STM.MASS=STM.MASS-STME.MASS;
      COST = 0.0;

```

Save the exit flows.

```
FLC1 = STM;
FLC2=STME;
RETURN;
```

```
STIN: ENTRY(ST_P);
```

STIN is an optional entry that may be called to obtain tables of exhaust loss vs. normalized mass flow rates. The data organization for a call to TABIN is described under TABIN.

```
IF MODE='DESIGN' THEN
  CALL TABIN(E_LOSS,DDNAME);
RETURN;

STOUT: ENTRY(ST_P);
PUT SKIP EDIT(' ',NAME)(COL(4),A);
PUT SKIP(2) EDIT('MODE = ',PARM.MODE,
'TURBINE EFFICIENCY = ',PARM.EFFICIENCY,
'MECHANICAL EFFICIENCY = ',PARM.MECH_EFF,
'POWER PRODUCED = ',POWER.PRODUCED,
'FLOW FACTOR = ',PARM.FLOW_FACT,
'DESIGN MASS FLOW RATE = ',PARM.DM,
'SPLIT RATIO = ',PARM.SR,
'VOL FLOW RATE = ',PARM.VOL_FLOW_RATE,
'EXHAUST LOSS = ',PARM.EXHAUST_LOSS)
(COL(10),A,A,8 (COL(10),A,E(13,5)));
IF INDEX(MODE,'IN')=0 THEN
  PUT EDIT('WHEEL SPEED = ',PARM.WHEEL_SPEED,
'WHEEL TO STEAM VEL RATIO = ',PARM.WV)
(COL(10),A,E(13,5));
END STC;
```

A.31 SYSTEM MODEL

A.31.1 Description of Model

The system model (SYST) calculates the total power put in, produced, consumed, and lost by the system. The model does not require any flows.

The parameters of the SYST model are as follows:

POWER_HEAD_PTR -- Specified pointer to the linked list of model POWER substructures.

FLOW_HEAD_PTR -- Specified pointer to the linked list of model FLOW substructures.

NET -- Calculated net power produced by the system (total power produced minus total power consumed).

EFFICIENCY -- Calculated system efficiency based on total power input, net power, and auxiliary power. If total input power is zero, EFFICIENCY is also set to zero.

AUXILIARY -- Specified value of any auxiliary-power requirements. AUXILIARY is subtracted from net power in calculating efficiency.

UNITS -- Character string that indicates SI for output in SI units; otherwise, British units are used for output.

A.31.2 Declaration Structure

```
* PROCESS NAME('SYSTC');
SYSTC: PROC(SYST_P);

DCL SYST_P POINTER;
DCL 1 SYST BASED(SYST_P),
    2 NAME CHAR(16),
    2 PARM,
    3 POWER_HEAD_PTR POINTER,
    3 FLOW_HEAD_PTR POINTER,
    3 NET FLOAT(16),
    3 EFFICIENCY FLOAT(16),
    3 AUXILIARY FLOAT(16),
    3 UNITS CHAR(7),
    2 SPOWER,
    3 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL 1 LINK BASED(LPT),
    2 (MP,LP,NP) POINTER;
DCL 1 POWER BASED(PPT),
    2 (INPUT,PRODUCED,CONSUMED,LOSS) FLOAT(16);
DCL 1 FLOW BASED(FPT),
    2 NAME CHAR(16),
    2 ID CHAR(16) VARYING,
    2 ATOM(8) FLOAT(16),
    2 PROP,
    3 (T,P,H,S,Q,R,V,M) FLOAT(16),
    2 COMP,
    3 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
        XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    2 SOL,
    3 WTF FLOAT(16);
DCL 1 MOD BASED,
    2 NAME CHAR(16);
DCL FNAME CHAR(16);
DCL MNAME CHAR(16);
DCL (LPT,PPT,FPT,TPT) POINTER;
DCL (NULL,ADDR) BUILTIN;
```

```

DCL Y(23) FLOAT(16) BASED(Y_PT);
DCL Y_PT POINTER;
DCL (I,J) FIXED BIN(15);
DCL FLAG BIT(1);
DCL (SV,E) FLOAT(16);
DCL SPEC(23) CHAR(3) STATIC INIT('AR','CH4','CO','CO2','H','H2',
    'H2O','H2S','K','KOH','NO','N2','O','OH','O2','SO2','HCL',
    'MET','C','COS','NH3','S','CL');
SPOWER=0.0;
DO LPT=POWER HEAD_PTR REPEAT(LINK.NP) WHILE(LPT=NULL);
    PPT=LINK.LP;
    SPOWER=SPOWER+POWER;
END;
NET=SPOWER.PRODUCED-SPOWER.CONSUMED-AUXILIARY;
IF SPOWER.INPUT=0.0 THEN
    EFFICIENCY=NET/SPOWER.INPUT;
ELSE
    EFFICIENCY=0.0;
RETURN;

SYSTOUT: ENTRY(SYST_P);
IF FLOW HEAD_PTR=NULL THEN
    IF FLOW_HEAD_PTR->LINK.MP=NULL THEN
        DO;
            PUT PAGE EDIT('OUTPUT BY FLOW')(COL(50),A);
            DO LPT=FLOW HEAD_PTR REPEAT(LINK.NP) WHILE(LPT=NULL);
                FPT=LINK.LP;
                FNAME=FLOW.NAME;
                DO TPT=FLOW HEAD_PTR REPEAT(TPT->LINK.NP) WHILE(TPT=LPT);
                    IF TPT->LINK.LP->FLOW.NAME=FNAME THEN
                        GO TO NEXT_FLOW;
                END;
                PUT EDIT('FLOW: ',FNAME)(SKIP(2),COL(2),A,A);
                PUT EDIT(' ', 'MODEL', 'PRES.', 'TEMP.', 'VELOCITY', 'ENTH.',
                    'MASS', 'SPEC VOL', 'ENERGY', 'QUALITY')
                    (SKIP,A,SKIP,COL(5),A,COL(28),A,COL(41),A,COL(51),A,COL(64),
                    A,COL(76),A,COL(86),A,COL(100),A,COL(110),A);
                IF UNITS='SI' THEN
                    PUT SKIP EDIT('(ATM)', '(K)', '(M/S)', '(J/KG)', '(KG/S)',
                        '(M**3/KG)', '(W)')
                        (COL(28),A,COL(42),A,COL(53),A,COL(63),A,COL(75),A,
                        COL(86),A,COL(101),A);
                ELSE
                    PUT SKIP EDIT('(PSI)', '(F)', '(FT/S)', '(BTU/LB)', '(LB/HR)',
                        '(FT**3/LB)', '(BTU/HR)')
                        (COL(28),A,COL(42),A,COL(52),A,COL(62),A,COL(75),A,
                        COL(86),A,COL(99),A);
                DO TPT=FLOW HEAD_PTR REPEAT(TPT->LINK.NP) WHILE(TPT=NULL);
                    FPT=TPT->LINK.LP;
                    IF FLOW.NAME=FNAME THEN
                        DO;
                            MNAME=TPT->LINK.MP->MOD.NAME;
                            SV=1.0/R;
                            E=M*(H+0.5*V*V);

```

```

      IF UNITS='SI' THEN
        PUT SKIP EDIT(MNAME,P,T,V,H,M,SV,E,Q)
          (COL(5),A,COL(25),7 (E(10,3),X(2)),E(8,1));
      ELSE
        PUT SKIP EDIT(MNAME,P*14.69595,1.8*T-459.67,V*3.2808,
          H/2324.444,M/1.25998E-4,SV*3.28**3/2.2,E/0.292875,Q)
          (COL(5),A,COL(25),7 (E(10,3),X(2)),E(8,1));
      END;
    END;
  NEXT_FLOW:
  END;
END;
FLAG='1'B;
IF FLOW_HEAD_PTR=NULL THEN
  IF FLOW_HEAD_PTR->LINK.MP=NULL THEN
    DO;
      DO LPT=FLOW_HEAD_PTR REPEAT(LINK.NP) WHILE(LPT=NULL);
      FPT=LINK.LP;
      IF FLOW.ID='GAS' THEN
        GO TO NEXT_FLOW_COMP;
      IF FLAG THEN
        DO;
          FLAG='0'B;
          PUT PAGE EDIT('COMPOSITION OUTPUT BY FLOW')(COL(40),A);
        END;
      FNAME=FLOW.NAME;
      DO TPT=FLOW_HEAD_PTR REPEAT(TPT->LINK.NP) WHILE(TPT=LPT);
      IF TPT->LINK.LP->FLOW.NAME=FNAME THEN
        GO TO NEXT_FLOW_COMP;
      END;
      PUT EDIT('FLOW: ',FNAME,' ')(SKIP(2),COL(2),A,A);
      DO TPT=FLOW_HEAD_PTR REPEAT(TPT->LINK.NP)
        WHILE(TPT=NULL);
      FPT=TPT->LINK.LP;
      IF FLOW.NAME=FNAME THEN
        DO;
          Y_PT=ADDR(FLOW.COMP);
          MNAME=TPT->LINK.MP->MOD.NAME;
          PUT SKIP EDIT(MNAME)(COL(5),A);
          I=0;
          DO J=1 TO 23;
            IF Y(J)>5E-6 THEN
              DO;
                I=I+1;
                IF I=8 | I=15 | I=22 THEN
                  PUT SKIP EDIT(' ')(COL(20),A);
                PUT EDIT(SPEC(J),'= ',Y(J),' ')(
                  A(3),A,F(7,5),A(2));
              END;
            END;
          END;
        END;
      END;
    NEXT_FLOW_COMP:
  END;

```

```

END;
IF POWER_HEAD_PTR =NULL THEN
  IF POWER_HEAD_PTR->LINK.MP=NULL THEN
    DO;
      PUT PAGE EDIT(' ','POWER SUMMARY')(COL(25),A);
      PUT SKIP EDIT('MODEL', 'INPUT', 'PRODUCED', 'CONSUMED',
        'LOSS')
        (SKIP,COL(5),A,COL(27),A,COL(38),A,COL(50),A,
        COL(63),A);
      PUT SKIP EDIT('(W)', '(W)', '(W)', '(W)', ' ')
        (COL(28),A,COL(40),A,COL(52),A,COL(63),A);
      DO LPT=POWER_HEAD_PTR REPEAT(LINK.NP) WHILE(LPT=NULL);
        PPT=LINK.LP;
        MNAME=LINK.MP->MOD.NAME;
        PUT SKIP EDIT(MNAME,POWER)
          (COL(5),A(15),COL(23),4 (X(2),E(10,3)));
      END;
      PUT EDIT(SYST.NAME,SPOWER)
        (SKIP(2),COL(5),A(15),COL(23),4 (X(2),E(10,3)));
      PUT SKIP EDIT('NET',SYST.NET,'AUXILIARY',SYST.AUXILIARY,
        'EFFICIENCY',SYST.EFFICIENCY)
        (SKIP,COL(5),A(15),COL(25),E(10,3));
    END;
  END SYSTC;

```

A.32 TWO-PHASE DIFFUSER MODEL

A.32.1 Description of Model

The two-phase, two-component diffuser model (TPDF) requires two pass-through flows, the first representing the gaseous-phase component and the second the liquid-phase component. Both of these flows are of the generic type GAS or LIQ.

The parameters of the TPDF model are as follows:

MODE -- Specified character string taking on the values " " or "SPEC-EFF." If "SPEC-EFF" is not set, then the efficiency of the diffusion process is calculated within the code (based on the void fraction of the flow).

EXIT_VELOCITY -- Specified exit velocity of the liquid flow.

SLIP_RATIO -- Specified ratio of the gas velocity to the liquid velocity.

EFFICIENCY -- Specified efficiency of the diffuser, defined as the ratio of change in pressure across the diffuser to the change in velocity head across the diffuser.

LENGTH -- Specified length of the diffuser (used in calculating additional pressure changes due to gravity).

GRAV_ANGLE -- Specified angle the diffuser makes with the gravitational field.

VOID_FRAC_IN -- Calculated inlet void fraction.

VOID_FRAC_OUT -- Calculated exit void fraction.

A.32.2 Declaration Structure

```
* PROCESS NAME('TPDFC');
TPDFC: PROC(TPDF_P, GAS_P, LIQ_P);

DCL (TPDF_P, GAS_P, LIQ_P) POINTER;
DCL 1 TPDF BASED(TPDF_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 MODE CHAR(8),
    3 EXIT_VELOCITY FLOAT(16),
    3 SLIP_RATIO FLOAT(16),
    3 EFFICIENCY FLOAT(16),
    3 LENGTH FLOAT(16),
    3 GRAV_ANGLE FLOAT(16),
    3 VOID_FRAC_IN FLOAT(16),
    3 VOID_FRAC_OUT FLOAT(16);
DCL (H_EXIT,F,HL,HV) FLOAT(16);
DCL (R_IN,R_OUT,R_AV,GRAV) FLOAT(16);
```

```

DCL (I,J) FIXED BIN(15);
DCL GP ENTRY;
DCL SOV ENTRY( FLOAT(16), FLOAT(16), FIXED BIN(15),
  FLOAT(16), FLOAT(16), FIXED BIN(15), FIXED BIN(15), CHAR(8));
DCL GAS BASED(GAS_P) LIKE FLC1;
DCL LIQ BASED(LIQ_P) LIKE FLC2;

```

Save the inlet flows.

```

FLC1=GAS;
FLC2=LIQ;

```

Calculate the inlet void fraction.

```

VOID_FRAC_IN=1./((1.+LIQ.MASS*GAS.RHO*GAS.VEL/
  (GAS.MASS*LIQ.RHO*LIQ.VEL)));

```

Calculate the inlet density of the gas-liquid mixture.

```

R_IN=VOID_FRAC_IN*GAS.RHO + (1.-VOID_FRAC_IN)*LIQ.RHO;

```

Initially, set the average density equal to the inlet density.

```

R_AV = R_IN;

```

If the mode is not "SPEC-EFF," calculate the efficiency based on the inlet void fraction. A simple empirical fit is used.

```

IF MODE='SPEC-EFF' THEN
  PARM.EFFICIENCY=0.85-0.09*(VOID_FRAC_IN/(1.-VOID_FRAC_IN));

```

Set the exit velocities of both flows, based on the specified exit velocity and the specified slip ratio.

```

FLC2.VEL =PARM.EXIT_VELOCITY;
FLC1.VEL =PARM.EXIT_VELOCITY*PARM.SLIP_RATIO;

```

Calculate the exit enthalpy of the gas-liquid mixture, using conservation of energy and the change in velocities.

```

GRAV=9.80665*LENGTH*COSD(GRAV_ANGLE);
H_EXIT =GAS.MASS*(GAS.ENTH+0.5*(GAS.VEL**2-FLC1.VEL**2)+GRAV
  +LIQ.MASS*(LIQ.ENTH+0.5*(LIQ.VEL**2-FLC2.VEL**2)+GRAV);

```

When the gravitational angle is not 90°, the change in pressure across the diffuser is also dependent on the density change, which is not known. A simple corrector iteration is made to evaluate the average density along the diffuser. The calculations of the exit conditions are made twice, once with the average density set equal to the inlet value and once with the average density averaged over the inlet and exit values.

```

DO J=1, 2 WHILE(GRAV=0.0);

```

Calculate the exit pressure, based on the specified efficiency and the average density.

```
FLC2.PRES = LIQ.PRES + 0.5*LIQ.RHO/101325.*PARM.EFFICIENCY
          *(LIQ.VEL**2-FLC2.VEL**2)+R_AV*GRAV/101325.0;
```

The enthalpies of the individual fluids are obtained by iterating over the enthalpy of the gas flow. At each iteration, the enthalpy of the gas is used (through the properties code) to determine its temperature. The liquid temperature is then set equal to the gas temperature, and the properties code is called again to determine the liquid's enthalpy. The combined enthalpy is then obtained and used by the one-dimensional equation solver to determine the next iterative value for the gas enthalpy.

```
FLC1.PRES=FLC2.PRES;
DO I=1 TO 20;
  CALL GP(NAME,FLC1,10B);
  FLC2.TEMP=FLC1.TEMP;
  CALL GP(NAME,FLC2,1B);
  F=H_EXIT-FLC1.MASS*FLC1.ENTH-FLC2.MASS*FLC2.ENTH;
  CALL SOV(FLC1.ENTH,F,I,1E3,1E-5,20,0,'TPDF');
END;
```

Calculate the exit void fraction and exit density.

```
VOID_FRAC_OUT=1./(1.+LIQ.MASS*GAS.RHO*GAS.VEL/
  (GAS.MASS*LIQ.RHO*LIQ.VEL));
R_OUT=VOID_FRAC_OUT*GAS.RHO+(1.0-VOID_FRAC_OUT)*LIQ.RHO;
```

Bring the average density up to date so it can be used in the second iteration (if GRAV is not zero).

```
R_AV= (R_IN + R_OUT)/2.;
END;
```

Save the exit flows.

```
GAS = FLC1;
LIQ = FLC2;
RETURN;
TPDFOUT: ENTRY( TPDF P );
  PUT SKIP EDIT(' ',NAME)(COL(4),A)
    ('MODE=',MODE)(COL(10),A,A)
    ('LENGTH =',LENGTH,'GRAV ANGLE =',GRAV_ANGLE,
    'Efficiency =',PARM.EFFICIENCY,
    'SLIP RATIO =',PARM.SLIP_RATIO,
    'VOID FRAC_IN=',PARM.VOID_FRAC_IN,
    'VOID FRAC_OUT=',PARM.VOID_FRAC_OUT)
    (COL(10),A,E(12,5));
END TPDFC;
```

A.33 TWO-PHASE MIXER MODEL

A.33.1 Description of Model

The two-phase, two-component mixer model (TPMX) requires two pass-through flows, the first being the gaseous component and the second, the liquid component. The parameters of the TPMX model are as follows:

PRES_OUT_OPTION -- Specified character string defining an option for calculating the exit pressure as a weighted average of the gas and liquid inlet pressures. If mix is equal to "MIX," this option is used; otherwise, the output flow pressure is taken as the minimum inlet flow pressure minus any specified pressure drop.

PRES_DROP -- Specified pressure drop through the mixer.

DP_FRAC -- Specified fraction of the minimum or weighted average inlet pressure, used as an additional pressure drop through the mixer.

SLIP_RATIO -- Specified ratio between the gas and liquid flow velocities.

TEMP_DIFF -- Specified difference between the gas temperature and that of the liquid.

PRES_DIFF IN -- Calculated difference between the gas inlet pressure and that of the liquid.

VOID_FRACTION -- Calculated void fraction of the exit flow.

A.33.2 Declaration Structure

```
* PROCESS NAME('TPMXC');
TPMXC: PROC( TPMX_P, GAS_P, LIQ_P);

DCL (TPMX_P,GAS_P,LIQ_P) POINTER;
DCL 1 TPMX BASED(TPMX_P),
    2 NAME CHAR(16),
    2 FLCL,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
```

```

2 FLC2,
3 FNAME CHAR(16),
3 ID CHAR(16) VARYING,
3 ATOM(8) FLOAT(16),
3 PROP,
4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
3 COMP,
4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
3 SOL,
4 WTF FLOAT(16),
2 PARM,
3 PRES_OUT_OPTION CHAR(3),
3 PRES_DROP FLOAT(16),
3 DP_FRAC FLOAT(16),
3 SLIP_RATIO FLOAT(16),
3 TEMP_DIFF FLOAT(16),
3 PRES_DIFF_IN FLOAT(16),
3 VOID_FRACTION FLOAT(16);
DCL (H_EXIT,PRES_COM,F) FLOAT(16);
DCL GP_ENTRY;
DCL GAS BASED(GAS_P) LIKE FLC1;
DCL LIQ BASED(LIQ_P) LIKE FLC2;
DCL (ABS, MAX, MIN) BUILTIN;
DCL SOV ENTRY( FLOAT(16), FLOAT(16), FIXED BIN(15),
FLOAT(16), FLOAT(16), FIXED BIN(15), FIXED BIN(15), CHAR(8));
DCL I FIXED BIN(15);

```

Save the inlet flows.

```

FLC1 = GAS;
FLC2 = LIQ;

```

Calculate the exit pressure of both flows, based on the pressure option, PRES_OUT_OPTION; the fractional pressure drop, DP_FRAC; and any specified pressure drop, PRES_DROP.

```

PRES_DIFF_IN=GAS.PRES-LIQ.PRES;
PRES_COM=MIN(GAS.PRES,LIQ.PRES);
IF PRES_OUT_OPTION='MIX' THEN
  PRES_COM=(1E2*PRES_COM+MAX(GAS.PRES,LIQ.PRES))/101E0;
FLC1.PRES =(1.-PARM.DP_FRAC)*PRES_COM-PARM.PRES_DROP;
FLC2.PRES = FLC1.PRES;

```

Calculate the exit gas velocity, based on the specified slip ratio.

```

FLC1.VEL =PARM.SLIP_RATIO*LIQ.VEL;
FLC2.VEL =LIQ.VEL;

```

Calculate the exit enthalpy of the gas-liquid mixture, based on conservation of energy and the change in velocities.

```

H_EXIT =GAS.MASS * (GAS.ENTH+0.5*(GAS.VEL**2-FLC1.VEL**2))
+LIQ.MASS * (LIQ.ENTH+0.5*(LIQ.VEL**2-FLC2.VEL**2));

```

Determine the enthalpies of the gas and liquid by iterating over the gas enthalpy. Given a gas enthalpy value, the temperature can be obtained from the properties code. The liquid temperature is obtained from the specified temperature difference, and then the liquid enthalpy is obtained. When the combined mixture's enthalpy is equal to the previously calculated exit enthalpy, the iterations are stopped.

```

DO I=1 TO 20;
  CALL GP(NAME,FLC1,10B);
  FLC2.TEMP=FLC1.TEMP-PARM.TEMP_DIFF;
  CALL GP(NAME,FLC2,1B);
  F=H_EXIT-FLC1.MASS*FLC1.ENTH-FLC2.MASS*FLC2.ENTH;
  CALL SOV(FLC1.ENTH,F,I,1E3,1E-3,20,0,'TPMX');
END;

```

Calculate the exit void fraction for printout.

```

VOID_FRACTION=1./(1.+FLC2.MASS*FLC1.RHO*FLC1.VEL/
(FLC1.MASS*FLC2.RHO*FLC2.VEL));

```

Save the exit flows.

```

GAS=FLC1;
LIQ=FLC2;
RETURN;
TPMXOUT: ENTRY( TPMX P );
PUT SKIP EDIT(' ',NAME)(COL(4),A)
('void frac. =',VOID_FRACTION,
'SLIP RATIO =',PARM.SLIP_RATIO,'TEMP DIFF =',PARM.TEMP_DIFF,
'PRES DROP =',PARM.PRES_DROP,'DP_FRAC =',PARM.DP_FRAC)
(COL(10),A,E(12,5));
IF ABS(PARM.PRES_DIFF_IN)>0.1 THEN
PUT EDIT('**** WARNING ****','INLET PRES DIFF =',PARM.PRES_DIFF_IN)
(COL(10),A,COL(40),A,E(10,3));
END TPMXC;

```

A.34 TWO-PHASE NOZZLE MODEL

A.34.1 Description of Model

The two-phase, two-component nozzle model (TPNZ) requires two pass-through flows of the generic types GAS and LIQ. The first flow represents the gaseous component; the second, the liquid component. The parameters of the TPNZ model are as follows:

EFFICIENCY -- Specified efficiency of the nozzle, defined as the ratio of the actual change in enthalpy across the nozzle to the isentropic enthalpy change.

EXIT_PRES -- Specified exit pressure from the nozzle.

SLIP_RATIO -- Specified ratio of the gas velocity to the liquid velocity at the nozzle exit.

TEMP_DIFF -- Specified difference in temperature between the liquid and the gas at the nozzle exit.

LENGTH -- Specified length of the nozzle.

GRAV_ANGLE -- Specified angle between the nozzle and the gravitational field.

VOID_FRACTION -- Calculated void fraction of the flow at the nozzle exit.

A.34.2 Declaration Structure

```
* PROCESS NAME('TPNZC');
TPNZC: PROC( TPNZ_P, GAS_P, LIQ_P);

DCL (TPNZ_P, GAS_P, LIQ_P) POINTER;
DCL 1 TPNZ BASED(TPNZ_P),
    2 NAME CHAR(16),
    2 FLC1,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 FLC2,
    3 FNAME CHAR(16),
    3 ID CHAR(16) VARYING,
    3 ATOM(8) FLOAT(16),
    3 PROP,
    4 (TEMP,PRES,ENTH,ENTP,QUAL,RHO,VEL,MASS) FLOAT(16),
    3 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    3 SOL,
    4 WTF FLOAT(16),
    2 PARM,
    3 EFFICIENCY FLOAT(16),*
    3 EXIT_PRES FLOAT(16),
    3 SLIP_RATIO FLOAT(16),
```

```

      3 TEMP_DIFF FLOAT(16),
      3 LENGTH FLOAT(16),
      3 GRAV_ANGLE FLOAT(16),
      3 VOID_FRACTION FLOAT(16);
DCL (F,S_TOTAL,H_EXIT,H_AVAILABLE,PWORK) FLOAT(16);

DCL GAS_BASED(GAS_P) LIKE FLC1;
DCL LIQ_BASED(LIQ_P) LIKE FLC2;
DCL GP_ENTRY;
DCL SOV_ENTRY( FLOAT(16), FLOAT(16), FIXED BIN(15),
  FLOAT(16), FLOAT(16), FIXED BIN(15), FIXED BIN(15), CHAR(8));
DCL I FIXED BIN(15);

```

Save the inlet flows.

```

FLC1=GAS;
FLC2=LIQ;

```

Set the exit pressure to the specified value.

```

GAS.PRES =PARM.EXIT_PRES;
LIQ.PRES =PARM.EXIT_PRES;

```

Calculate the gas-liquid mixture's entropy.

```

S_TOTAL =GAS.MASS*GAS.ENTP+LIQ.MASS*LIQ.ENTP;

```

Determine the conditions of the flows after an isentropic expansion to the exit pressure by iterating over the gas enthalpy. For each iteration, given a value for the gas enthalpy, the gas temperature is obtained. The liquid temperature is then obtained from the specified liquid-gas temperature difference. The properties code is then called for the liquid to determine its entropy. The combined mixture's entropy is calculated, and the one-dimensional equation solver is then used to give the next iterative value of the gas enthalpy to force the calculated mixture entropy to equal the inlet mixture entropy.

```

DO I=1 TO 21;
  CALL GP(NAME,GAS,10B);
  LIQ.TEMP=GAS.TEMP+PARM.TEMP_DIFF;
  CALL GP(NAME,LIQ,1B);
  F=GAS.MASS*GAS.ENTP+LIQ.MASS*LIQ.ENTP-S_TOTAL;
  CALL SOV(GAS.ENTH,F,I,1E3,1E-3,21,0,'TPNZ1');
END;

```

The total available enthalpy change can be obtained by subtracting the isentropic mixture enthalpy from the inlet mixture enthalpy.

```

H_AVAILABLE =GAS.MASS*(FLC1.ENTH-GAS.ENTH)
  +LIQ.MASS*(FLC2.ENTH-LIQ.ENTH);

```

Using the specified efficiency of the nozzle, the mixture exit enthalpy is determined.

```
H_EXIT =FLC1.MASS*FLC1.ENTH+FLC2.MASS*FLC2.ENTH
        -PARAM.EFFICIENCY*H_AVAILABLE;
```

The exit enthalpies of the individual gas and liquid flows can be obtained in much the same way that the isentropic conditions were obtained. An iteration is performed over the gas enthalpy. Using the properties code, the gas temperature, liquid temperature, and liquid enthalpy are obtained. The mixture enthalpy is then determined and used by the one-dimensional equation solver to determine the next iterative value of the gas enthalpy necessary to force the mixture enthalpy to equal the exit enthalpy.

```
DO I=1 TO 21;
    CALL GP(NAME,GAS,10B);
    LIQ.TEMP=GAS.TEMP+PARAM.TEMP_DIFF;
    CALL GP(NAME,LIQ,1B);
    F=GAS.MASS*GAS.ENTH+LIQ.MASS*LIQ.ENTH-H_EXIT;
    CALL SOV(GAS.ENTH,F,I,1E3,1E-3,21,0,'TPNZ2');
END;
```

The liquid exit velocity is determined, based on conservation of energy. The total amount of energy that can be converted to a velocity head is determined from the actual enthalpy change and from any additional potential energy change due to the gravitational field.

```
PWORK =H_AVAILABLE*PARAM.EFFICIENCY
        +(GAS.MASS+LIQ.MASS)*9.80665*LENGTH*COSD(GRAV_ANGLE)
        +.5*(GAS.MASS*GAS.VEL**2
        +LIQ.MASS*LIQ.VEL**2);
```

If PWORK is very small, then for safety the liquid velocity is simply assigned a small value. Otherwise, the liquid velocity is calculated such that the combined gas and liquid velocity head is equal to PWORK. (The specified slip ratio is also used here to determine the gas velocity in terms of the liquid velocity.)

```
IF PWORK < 1E-6 THEN
    LIQ.VEL=1E-6;
ELSE
    LIQ.VEL=SQRT(2.*PWORK/(GAS.MASS*PARAM.SLIP_RATIO**2+LIQ.MASS));
```

The gas velocity is set, based on the slip ratio.

```
GAS.VEL =PARAM.SLIP_RATIO*LIQ.VEL;
```

The void fraction is calculated for output.

```
VOID_FRACTION=1./(1.+LIQ.MASS*GAS.RHO*GAS.VEL/
    (GAS.MASS*LIQ.RHO*LIQ.VEL));
```

The exit flow conditions are saved.

```

FLC1=GAS;
FLC2=LIQ;
RETURN;
TPNZOUT: ENTRY( TPNZ P );
PUT SKIP EDIT(' ',NAME)(COL(4),A)
('Efficiency =',PARM.EFFICIENCY,'VOID FRAC =',VOID_FRACTION,
'SLIP RATIO=',SLIP_RATIO,'TEMP DIFF=',TEMP_DIFF,
'LENGTH =',LENGTH,'GRAV ANGLE =',GRAV_ANGLE)
(COL(10),A,E(12,5));
END TPNZC;

```

A.35 GENERAL PROPERTIES CODE

```

* PROCESS NAME('GPIN');
GPIN: PROC(GP_P);

DCL GP_P POINTER;
DCL 1 GP_BASED(GP_P),
    2 NAME_CHAR(16),
    2 PARM,
    3 PRINT_FIXED_BIN(15),
    1 FLOW_CONN,
    2 NAME_CHAR(16),
    2 ID_CHAR(16) VARYING,
    2 ATOM(8) FLOAT(16),
    2 PROP,
    4 (T,P,H,S,Q,R,V,M) FLOAT(16),
    2 COMP,
    4 (XAR,XCH4,XCO,XCO2,XH,XH2,XH2O,XH2S,XK,XKOH,XNO,XN2,
      XO,XOH,XO2,XSO2,XHCL,XCH3OH,XC,XCOS,XNH3,XS,XCL) FLOAT(16),
    2 SOL,
    4 WTF_FLOAT(16),
    1 MAT_BASED(M_PTR),
    2 ID_CHAR(16) VARYING,
    2 (PTR,P_PTR) POINTER,
    (NULL,SUBSTR,INDEX) BUILTIN,
    (GASIN,GASWK) ENTRY,
    (STMHP,STMSP,STMT_P,STMSAT) ENTRY,
    (THRIN,THRHP,THRSP,THRTP,THRSAT) ENTRY,
    (SODHP,SODTP,SODSAT) ENTRY,
    (POTHP,POTTP,POTSAT) ENTRY,
    (LIQIN,LIQWK) ENTRY,
    (JANIN,JANWK) ENTRY,
    (THR_PTR,JAN_PTR,LIQ_PTR) POINTER_INIT(NULL) STATIC,
    GPPRT_FIXED_BIN(15) EXT,
    LABEL_CHAR(16),
    CODETYPE_CHAR(16) VARYING,
    M_PTR POINTER,
    (HL,HS,PCRT,D1,D2,D3,D4,TSAT) FLOAT(16),
    (I,SW) FIXED_BIN(15);

```

```

CALL GASIN;
GPPRT=GP.PRINT;
RETURN;
GP: ENTRY(LABEL,FLOW,SW);
CODETYPE=SUBSTR(FLOW.ID,1,INDEX(FLOW.ID||' ','-')-1);
SELECT(CODETYPE);
  WHEN('H2O')
    DO;
      FLOW.Q=0.0;
      SELECT(SW);
        WHEN(1) CALL STMP(FLOW.T,FLOW.P,FLOW.H,FLOW.S,FLOW.R);
        WHEN(2) CALL STMHP(FLOW.H,FLOW.P,FLOW.T,FLOW.S,FLOW.R,FLOW.Q);
        WHEN(3) CALL STMSP(FLOW.S,FLOW.P,FLOW.T,FLOW.H,FLOW.R,FLOW.Q);
      END;
    END;
  WHEN('GAS')
    DO;
      Q=1.0;
      CALL GASWK(LABEL,FLOW.COMP,FLOW.ATOM,FLOW.T,FLOW.P,FLOW.H,
        FLOW.S,FLOW.R,FLOW.WTF,SW);
    END;
  WHEN('LIQ')
    DO;
      DO M_PTR=LIQ_PTR REPEAT(MAT.P_PTR) WHILE(M_PTR=NULL);
      IF FLOW.ID=MAT.ID THEN
        GO TO LIQCAL;
      END;
      ALLOC MAT SET(M_PTR);
      MAT.P_PTR=LIQ_PTR;
      LIQ_PTR=M_PTR;
      MAT.ID=FLOW.ID;
      CALL LIQIN(MAT.PTR,MAT.ID);
    LIQCAL:
      CALL LIQWK(MAT.PTR,FLOW.T,FLOW.P,FLOW.H,FLOW.S,FLOW.R,FLOW.Q,SW);
    END;
  WHEN('JAN')
    DO;
      DO M_PTR=JAN_PTR REPEAT(MAT.P_PTR) WHILE(M_PTR=NULL);
      IF FLOW.ID=MAT.ID THEN
        GO TO JANCAL;
      END;
      ALLOC MAT SET(M_PTR);
      MAT.P_PTR=JAN_PTR;
      JAN_PTR=M_PTR;
      MAT.ID=FLOW.ID;
      CALL JANIN(MAT.PTR,MAT.ID);
    JANCAL:
      CALL JANWK(MAT.PTR,FLOW.T,FLOW.P,FLOW.H,FLOW.S,FLOW.R,FLOW.Q,SW);
    END;
  WHEN('THR')
    DO;
      FLOW.Q=0;
      DO M_PTR=THR_PTR REPEAT(MAT.P_PTR) WHILE(M_PTR=NULL);
      IF FLOW.ID=MAT.ID THEN

```

```

      GO TO THRCAL;
END;
ALLOC MAT SET(M PTR);
MAT.P PTR=THR PTR;
THR PTR=M PTR;
MAT.ID=FLOW.ID;
CALL THRIN(MAT.PTR,MAT.ID);
THRCAL:
SELECT(SW);
  WHEN(1) CALL THRTP(MAT.PTR,FLOW.T,FLOW.P,FLOW.R,FLOW.H,FLOW.S,
    D1,D2,D3);
  WHEN(2) CALL THRHP(MAT.PTR,FLOW.H,FLOW.P,FLOW.T,FLOW.R,
    FLOW.S,FLOW.Q);
  WHEN(3) CALL THRSP(MAT.PTR,FLOW.S,FLOW.P,FLOW.T,FLOW.R,
    FLOW.H,FLOW.Q);
END;
END;
WHEN('SOD')
DO;
  SELECT(SW);
    WHEN(1) CALL SODTP(FLOW.T,FLOW.P,FLOW.R,FLOW.H,FLOW.S,
      D1,D2,D3);
    WHEN(2) CALL SODHP(FLOW.H,FLOW.P,FLOW.T,FLOW.R,
      FLOW.S,FLOW.Q);
    OTHERWISE
      PUT SKIP EDIT(' OPTION NOT AVAILABLE FOR ',FLOW.ID)
        (A,A);
  END;
END;
WHEN('POT')
DO;
  SELECT(SW);
    WHEN(1) CALL POTTP(FLOW.T,FLOW.P,FLOW.R,FLOW.H,FLOW.S,
      D1,D2,D3);
    WHEN(2) CALL POTHP(FLOW.H,FLOW.P,FLOW.T,FLOW.R,
      FLOW.S,FLOW.Q);
    OTHERWISE
      PUT SKIP EDIT(' OPTION NOT AVAILABLE FOR ',FLOW.ID)
        (A,A);
  END;
END;
OTHERWISE
DO;
  PUT EDIT(' UNRECOGNIZED FLOW TYPE: ',FLOW.NAME,' USED AT ',LABEL)
    (SKIP,A,A,A,A);
  STOP;
END;
END;
IF GPRT>0 THEN
  PUT EDIT(LABEL,FLOW.NAME,FLOW.ID,'T=',FLOW.T,'P=',FLOW.P,
    'H=',FLOW.H,'S=',FLOW.S,'M=',FLOW.M,'SW=',SW)
    (COL(2),A(16),A(16),A(4),X(2),5 (A(2),E(11,4),X(2)),A,F(2));
RETURN;
GPSAT: ENTRY(LABEL,FLOW,PCRT,HL,HS);

```

```

CODETYPE=SUBSTR(FLOW.ID,1,INDEX(FLOW.ID||' ','-')-1);
SELECT(CODETYPE);
  WHEN('H2O') CALL STMSAT(FLOW.P,PCRIT,HL,HS);
  WHEN('THR')
    DO;
      DO M_PTR=THR_PTR REPEAT(MAT.P_PTR) WHILE(M_PTR=NULL);
        IF FLOW.ID=MAT.ID THEN
          GO TO THRSATCAL;
        END;
      ALLOC MAT SET(M_PTR);
      MAT.P_PTR=THR_PTR;
      THR_PTR=M_PTR;
      MAT.ID=FLOW.ID;
      CALL THRIN(MAT.PTR,MAT.ID);
    THRSATCAL:
      CALL THRSAT(MAT.PTR,FLOW.P,PCRIT,TSAT);
      CALL THRTP(MAT.PTR,TSAT,FLOW.P,D1,HL,D2,D3,HS,D4);
    END;
  WHEN('SOD')
    DO;
      CALL SODSAT(FLOW.P,PCRIT,TSAT);
      CALL SODTP(TSAT,FLOW.P,D1,HL,D2,D3,HS,D4);
    END;
  WHEN('POT')
    DO;
      CALL POTSAT(FLOW.P,PCRIT,TSAT);
      CALL POTTP(TSAT,FLOW.P,D1,HL,D2,D3,HS,D4);
    END;
  OTHERWISE
    DO;
      PUT EDIT(' NO SATURATION PROPERTIES FOR FLOW TYPE: ',FLOW.NAME,
        ' USED AT ',LABEL)
        (SKIP,A,A,A,A);
      STOP;
    END;
END;
END GPIN;

```


APPENDIX B: JOB-CONTROL LANGUAGE FOR IBM SYSTEM AT ANL

The preceding chapters (and Appendix A) have dealt with the data that must appear within the STRUCT file. This file is usually the only file that must be changed when running a new systems-analysis problem. However, other files are used by the SALT code in the process of compiling the PL/I driver that represents the system under consideration. Essentially, these other files are temporary work files or output files and are not usually saved from job to job.

Three major steps are required in running the SALT system code after the STRUCT file has been prepared. The first step is to run the SALT code itself and translate the STRUCT file into a PL/I code; the second step is to compile this code, and the third step is to execute the PL/I code. The performance of these three steps has been conveniently arranged in an instream job-control-language (JCL) procedure called SYSTEM for use on the ANL computer. The JCL using this procedure is as follows:

```
//JOBNAME JOB TIME=2,REGION=350K,CLASS=W,MSGCLASS=W
//*MAIN ORG=LOCAL,SYSTEM=(S33A,S33B),LINES=5
//SYSTEM PROC
//ONE EXEC PGM=SALT
//STEPLIB DD DSN=Bxxxxx.SALT.LOAD,DISP=SHR
//STRUCT DD DDNAME=STRUCIN
//INTF DD DSN=Bxxxxx.SALT.INTF,DISP=SHR
//SYSDRV DD UNIT=SASCR,SPACE=(TRK,(2,1)),DISP=(NEW,PASS)
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=1511)
//PLO EXEC PGM=IELOAA,PARM='NS,NA,NX,NAG,NOESD,NSTG,NOF,NOP'
//STEPLIB DD DSN=PLI.OPT.LINKLIB,DISP=SHR
//SYSIN DD DSN=*.ONE.SYSDRV,DISP=(OLD,DELETE)
//SYSLIN DD UNIT=SASCR,SPACE=(CYL,6),
// DISP=(NEW,PASS),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=1511)
//SYSPUNCH DD DUMMY
//SYSUT1 DD SPACE=(CYL,6),UNIT=(SASCR)
//TWO EXEC PGM=LOADER,REGION=150K,COND=(9,LT,PLO)
//SYSLIB DD DSN=SYS1.PLIBASE,DISP=SHR
// DD DSN=Bxxxxx.SALT.LOAD,DISP=SHR
//SYSLIN DD DSN=*.PLO.SYSLIN,DISP=(OLD,DELETE)
//SYSLOUT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=121,BLKSIZE=1573)
//SYSPNCH DD DUMMY
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=1511)
//SYSPUNCH DD DUMMY
// PEND
// EXEC SYSTEM
//ONE.STRUCIN DD *
```

(contents of file STRUCT)

These JCL lines, which carry out the three basic steps referred to above, are briefly described here.

The first line of any JCL, the JOB card, specifies the maximum time (in minutes) that a job is permitted to run on the computer and the amount of main core used. The parameter CLASS defines the priority of the job and may take the values of U (for highest priority), W (for normal priority), X (for overnight service), or Y (for weekend service). The MSGCLASS parameter may be set to W to fetch the output at the computer terminal or to A to print the output on a line printer.

The second line specifies the on-line printer destinations, the computer used, and the maximum number of output lines. If a large number of parameter sweeps are to be performed, the LINES parameter (which specifies the maximum number of lines in thousands) may need to be increased.

The next line specifies the beginning of the SYSTEM procedure. The following group of six lines carries out the first step, the translation of the STRUCT file. Here, STEPLIB is the data set containing the SALT code, INTF is the interface file, SYSDRV is the output file containing the generated PLI code, and SYSPRINT contains a reflection of the STRUCT file and possible error messages.

The next eight lines, starting with //PLO, accomplish the compilation of the PLI code. In this case, STEPLIB refers to the data set containing the PLI compiler, SYSIN is the PLI code generated in the first step, SYSLIN is the compiled code, SYSPRINT contains error messages from the compilation, SYSPUNCH is not used, and SYSUT1 is a work file used by the compiler.

The next group of eight lines carries out the final step required in running the compiled code. The SYSLIB file contains the concatenation of several data sets representing the components of the system and various IBM-supplied procedures, such as SIN, COS, ABS, etc. (These are in SYS1.PLIBASE for PLI codes). The component models and other mathematical procedures are referenced by the next two lines. Here, SYSLIN refers to the compiled PLI code, SYSLOUT contains the loader map and loader error messages, SYSPNCH and SYSPUNCH are not used, and SYSPRINT contains the major output for the system analysis.

Finally, the next two lines close the instream procedure (// PEND) and execute this procedure (// EXEC SYSTEM). For most systems-analysis problems, the above JCL need not be changed from job to job. The rest of the JCL represents the STRUCT file, preceded by the //ONE.STRUCIN DD * line.

Distribution for ANL/FE-85-4

Internal:

J.G. Asbury	S.J. Grammel	C.V. Pearson
F.C. Bennett	W. Harrison	M. Petrick
M.J. Bernard	J.E. Helt	G.N. Reddy
G.K. Berry (30)	D.R. Henley	J.J. Roberts
S.K. Bhattacharyya	H.S. Huang	N.F. Sather
D.J. Bingaman	J.F. Koenig	W.W. Schertz
L.W. Carlson	M. Krumpelt	R. Sekar
K.C. Chang	K.D. Kuczen	Y.W. Shin
S.U. Choi	J. Lazar	A.J. Sistino
L.S. Chow	C. Lee	T.G. Surles
J.M. Cook	G.P. Lewis	C.E. Swietlik
E.J. Croke	R.A. Lewis	A. Thomas
E.J. Daniels	C.D. Livengood	S.P. Vanka
E.M. Dean	R.W. Lyczkowski	C.S. Wang
C.B. Dennis	K.S. Macal	A.M. Wolsky
D.R. Diercks	V. Minkov	ANL Contract Copy
J.J. Dzingel	K.M. Myles	ANL Libraries (2)
H.K. Geyer (10)	O.O. Ohlsson	ANL Patent Department
R.F. Giese	C.B. Panchal	TIS Files (6)

External:

U.S. Department of Energy Technical Information Center, for distribution per UC-32 and UC-90 (242)

Manager, U.S. Department of Energy Chicago Operations Office (DOE-CH)

Energy and Environmental Systems Division Review Committee:

R.S. Berry, The University of Chicago

G.E. Dials, Dials and Assoc., Santa Fe, N.M.

B.A. Egan, Environmental Research and Technology, Inc., Concord, Mass.

W.H. Esselman, Electric Power Research Institute, Palo Alto, Calif.

M.H. Kohler, Bechtel National, Inc., San Francisco

J.W. McKie, University of Texas, Austin

N.C. Mullins, Virginia Polytechnic Institute and State University, Blacksburg

J.J. Stukel, University of Illinois, Urbana

J.J. Wortman, North Carolina State University, Raleigh

R.D. Andrews, Rocky Mountain Energy, Broomfield, Colo.

R. Bajura, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.

J.M. Begovich, Oak Ridge National Laboratory, Oak Ridge, Tenn.

S.K. Beer, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.

S. Biondo, Office of Fossil Energy, U.S. Department of Energy, Washington, D.C.

H.H. Blecker, ICARUS Corp., Rockville, Md.
 D.P. Bloomfield, PSI/Systems, Andover, Mass.
 A. Boni, Physical Sciences, Inc., Andover, Mass.
 H. Branover, Ben-Gurion University, Tel Aviv, Israel
 D. L. Breton, Dow Chemical USA, Plaquemine, La.
 R. Carabetta, Pittsburgh Energy Technology Center, U.S. Department of Energy,
 Pittsburgh
 P. Chung, University of Illinois, Chicago
 J.G. Cleland, Research Triangle Institute, Research Triangle Park, N.C.
 K. Craig, Morgantown Energy Technology Center, U.S. Department of Energy,
 Morgantown, W.Va.
 J. Cutting, Gilbert Associates, Reading, Penn.
 K.J. Daniel, General Electric Corporate R&D, Schenectady, N.Y.
 S. Divakaruni, Electric Power Research Institute, Palo Alto, Calif.
 J.S. Dweck, J.S. Dweck, Inc., Denver, Colo.
 A. Dyson, Tennessee Valley Authority, Chattanooga, Tenn.
 J. Elliott, Massachusetts Institute of Technology, Cambridge, Mass.
 G. Enyedy, PDQ\$, Inc., Gates Mills, Ohio
 M. Faist, Radian Corp., Austin, Texas
 L.T. Fan, Kansas State University
 J. Fillo, Environmental Research and Technology, Inc., Pittsburgh
 J. Fisher, Stone and Webster, Boston
 H.J. Gadiyar, Morgantown Energy Technology Center, U.S. Department of Energy,
 Morgantown, W.Va.
 P.W. Gallier, ASPEN Technology, Inc., Cambridge, Mass.
 G. Garrison, University of Tennessee Space Institute, Tullahoma, Tenn.
 E.W. Geller, Flow Industries, Inc., Kent, Wash.
 J.H. Gibbons, Office of Technology Assessment, U.S. Congress
 F.D. Gmeindl, Morgantown Energy Technology Center, U.S. Department of Energy,
 Morgantown, W.Va.
 L.E. Graham, Morgantown Energy Technology Center, U.S. Department of Energy,
 Morgantown W.Va.
 H. Hagler, Hagler, Bailly, and Co., Washington, D.C.
 K. Haynes, Foster Wheeler Synfuels Corp., Livingston, N.J.
 J. Henry, University of Tennessee at Chattanooga
 S.C. Hill, Los Alamos National Laboratory, Los Alamos, N.M.
 R. Holmann, Westinghouse Electric Corp., Pittsburgh
 F. Honea, Grand Forks Project Office, Grand Forks, N.D.
 D.A. Horazak, Westinghouse Electric Corp., Concordville, Penn.
 W. Jackson, HNJ Corp., Washington, D.C.
 B. Joseph, Washington University, St. Louis, Mo.
 D.E. Kash, University of Oklahoma
 A.A. Khan, Union Carbide Corp., Oak Ridge, Tenn.
 S. Knoke, Flow Industries, Inc., Kent, Wash.
 D. Krastman, Pittsburgh Energy Technology Center, U.S. Department of Energy,
 Pittsburgh
 H. Link, Solar Energy Research Institute, Golden, Colo.
 T. Littert, Westinghouse R&D, Pittsburgh

P.S. Lowell, P.S. Lowell Co., Inc., Austin, Texas
 C. Mah, Aerojet Energy Conversion Co., Sacramento, Calif.
 T. McCloskey, Notre Dame College, South Euclid, Ohio
 W.J. McMichael, Research Triangle Institute, Research Triangle Park, N.C.
 M.C. Millman, Halcon Computer Technology, New York
 L. Miller, U.S. Department of Energy, Germantown, Md.
 L. Mims, Chicago
 L.M. Naphtali, U.S. Department of Energy, Washington, D.C.
 S.A. Newman, Foster Wheeler Energy Corp., Livingston, N.J.
 J. Notestein, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.
 T. O'Brien, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.
 A. Pappano, Pasadena, Calif.
 M. Paskin, Allison Gas Turbine, Indianapolis, Ind.
 J. Patten, Gilbert Associates, Reading, Penn.
 L. Perini, Applied Physics Lab, Johns Hopkins Laboratory, Laurel, Md.
 M. Perlmutter, U.S. Department of Energy, Pittsburgh
 T.T. Philips, Los Alamos National Laboratory, Los Alamos, N.M.
 R. Piccirelli, Wayne State University, Detroit, Mich.
 E. Pierson, Purdue University-Calumet, Hammond, Ind.
 A.A. Pitrolo, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.
 P. Probert, Babcock and Wilcox Co., Barberton, Ohio
 G.H. Quentian, Electric Power Research Institute, Palo Alto, Calif.
 R. Raghavan, Foster Wheeler Energy Corp., Livingston, N.J.
 M.W. Reed, Tennessee Valley Authority, Chattanooga, Tenn.
 I.H. Rinard, Halcon SD Group, New York
 L. Saroff, Dravo Engineers, Inc., Pittsburgh
 R. Shinnar, City College of New York
 C.H. Sink, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.
 D.P. Smith, General Electric Corporate R&D, Schenectady, N.Y.
 I. Smith, The City University, London, U.K.
 G. Steinfeld, Science Applications, Inc., Morgantown, W.Va.
 S.S. Stern, Halcon SD Group, New York
 K. Stone, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.
 B. Svrcek, University of Calgary, Alberta
 D. Swink, Office of Fossil Energy, U.S. Department of Energy, Washington, D.C.
 J. Templemeyer, Southern Illinois University, Carbondale, Ill.
 W.C. Thomas, Radian Corp., Austin, Texas
 W. Trzaskoma, Gilbert Assoc., Inc., Reading, Penn.
 V.S. Underkoffler, Gilbert Assoc., Inc., Reading, Penn.
 S.R. Vatcha, Ashland Oil, Inc., Ashland, Ky.
 K. Vyas, Morgantown Energy Technology Center, U.S. Department of Energy, Morgantown, W.Va.
 R.E. Weinstein, Gilbert/Commonwealth, Reading, Penn.

- J. Weisman, University of Cincinnati, Ohio
 W. Wells, Center for Research on Sulfur in Coal, Champaign, Ill.
 G. Wheeler, U.S. Department of Energy, Germantown, Md.
 F. Wong, Electric Power Research Institute, Palo Alto, Calif.
 S. Wu, University of Tennessee Space Institute, Tullahoma, Tenn.
 R.K. Young, Stearns-Catalytic, Inc., Homer City, Penn.
 J. Zaranek, U.S. Steel Corp., Monroeville, Penn.

ARGONNE NATIONAL LAB WEST



3 4444 00008919 3

x

